

AD-A086 334

IBM FEDERAL SYSTEMS DIV GAITHERSBURG MD  
ANALYSIS OF DISCRETE SOFTWARE RELIABILITY MODELS.(U)  
APR 80 W D BROOKS, R W MOTLEY

F/G 9/2

F30602-78-C-0346

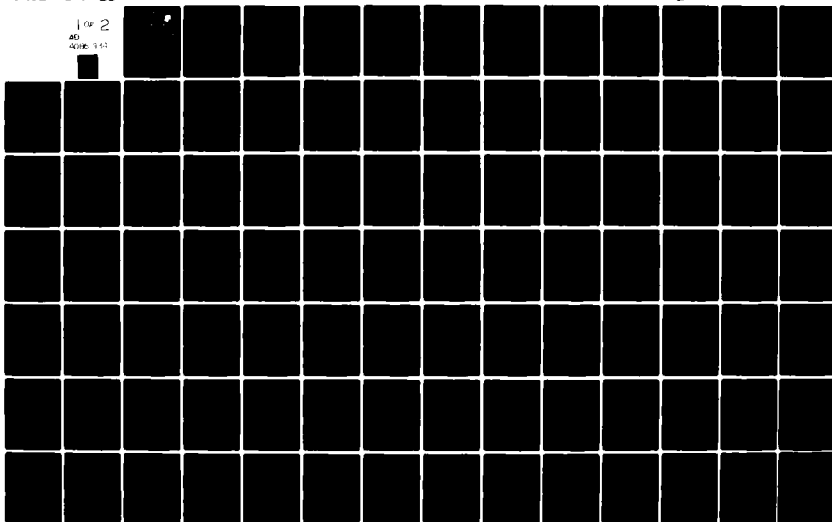
UNCLASSIFIED

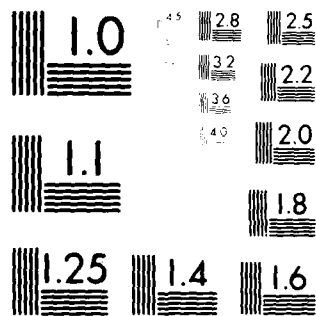
RADC-TR-80-84

NL

1 of 2

AD  
4000 1-1





MEADOWS RESOLUTION TEST CHART  
 NATIONAL BUREAU OF STANDARDS-1963-A

5  
RADC-TN-80-24  
Final Technical Report  
April 1980

**LEVEL**

(12)



## **ANALYSIS OF DISCRETE SOFTWARE RELIABILITY MODELS**

**IBM Corporation**

W. D. Brooks  
R. W. Motley

DTIC  
ELECTE  
JUL 8 1980  
S C D

APPROVED FOR PUBLIC RELEASE; DISTRIBUTION UNLIMITED

DDC FILE COPY AD A 086334

**ROME AIR DEVELOPMENT CENTER  
Air Force Systems Command  
Griffiss Air Force Base, New York 13441**

80 7 7 031

This report has been reviewed by the Public Affairs Office (PAO) and is releasable to the National Technical Information Service (NTIS). At present it will be releasable to the general public, including foreign nations.

RADC-TR-80-84 has been reviewed and is approved for publication.

APPROVED:

*Alan N. Sukert*

ALAN N. SUKERT  
Project Engineer

APPROVED:

*Wendall C. Bauman*

WENDALL C. BAUMAN, Colonel, USAF  
Chief, Information Sciences Division

FOR THE COMMANDER:

*John P. Huss*

JOHN P. HUSS  
Acting Chief, Plans Office

If your address has changed or if you wish to be removed from the RADC mailing list, or if the addressee is no longer employed by your organization, please notify RADC (ISIS), Griffiss AFB NY 13441. This will assist us in maintaining a current mailing list.

Do not return this copy. Retain or destroy.

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

19 REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM	
1. REPORT NUMBER RADC TR-86-84	2. GOVT ACCESSION NO. ADA086334	3. RECIPIENT'S CATALOG NUMBER	
4. TITLE (and Subtitle) ANALYSIS OF DISCRETE SOFTWARE RELIABILITY MODELS		5. SCOPE OF REPORT & PERIOD COVERED Final Technical Report, Oct 78 - Sep 79	
7. AUTHOR(s) W. D. Brooks R. W. Motley		6. PERFORMING ORG. REPORT NUMBER N/A	
9. PERFORMING ORGANIZATION NAME AND ADDRESS IBM Corporation/Federal Systems Division, 18100 Frederick Pike Gaithersburg MD 20760		8. CONTRACT OR GRANT NUMBER(s) F30602-78-C-0346	
11. CONTROLLING OFFICE NAME AND ADDRESS Rome Air Development Center (ISIS) Griffiss AFB NY 13441		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS 62702F 55812013	
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) Same		12. REPORT DATE Apr 1980	
(12) 144		13. NUMBER OF PAGES 140	
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		15. SECURITY CLASS. (of this report) UNCLASSIFIED	
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report) Same		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE N/A	
18. SUPPLEMENTARY NOTES RADC Project Engineer: Alan N. Sukert (ISIS)			
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) Software Reliability      Software Reliability Models Software Error Analysis      Software Quality and Error-Freeness			
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) Discrete (binomial and Poisson) software reliability models, previously developed by IBM, are presented. They were examined for their validity using historical data provided by RADC from seven development projects. Simulated error data that closely agreed with the model assumptions were also used for validation. The results indicated that the models provide reasonable fits to the historical error data. It is recommended that future studies of the models be accomplished with error data that are			

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

174950

Jm

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

properly collected according to the model data requirements and assumptions.

Accession For	
NTIS GRA&I	<input checked="checked" type="checkbox"/>
DOC TAB	<input type="checkbox"/>
Unannounced	<input type="checkbox"/>
Justification	
By	
Distribution/	
Availability	
Dist	Available and/or special
A	

UNCLASSIFIED

SECURITY CLASSIFICATION OF THIS PAGE(When Data Entered)

## CONTENTS

Section		Page
1	INTRODUCTION	1-1
1.1	Background and Purpose	1-2
1.2	Major Results and Conclusions Concerning Model Validity	1-3
2	THEORETICAL FOUNDATIONS AND MODEL DEVELOPMENT	2-1
2.1	Introduction and Overview	2-1
2.2	Problem Statement	2-3
2.3	Mathematical Formulation	2-7
2.3.1	Expected Value for the Binomial Distribution by Module	2-9
2.3.2	Expected Value for the Binomial Distribution for the System	2-12
2.3.3	Expected Value for the Poisson Distribution by Module	2-14
2.3.4	Expected Value for the Poisson Distribution for the System	2-15
2.4	Parameter Estimation	2-16
2.4.1	Binomial Maximum Likelihood	2-16
2.4.2	Poisson Maximum Likelihood	2-19
2.4.3	Least Squares	2-21
2.5	Solution Procedures	2-23
2.6	Correctness Verification	2-25
2.7	Initial Value and Convergence Considerations	2-26
3	MODEL ANALYSIS REQUIREMENTS AND RESULTS	3-1
3.1	Introduction	3-1
3.2	Model Validation	3-3
3.2.1	Historical Error Data Analyzed	3-3
3.2.2	Data Assumptions	3-7
3.2.3	Results	3-9
3.2.4	Discussion of Results	3-22
3.2.5	Conclusions Based on Analysis of Historical Data	3-25
3.2.6	Model Validation by Means of Simulation	3-27
3.3	Optimum Time Interval for Data Aggregation	3-35
3.4	Time to a Specified Number of Remaining Errors	3-37
3.4.1	Formulation	3-37
3.4.2	Example	3-41
3.4.3	Combining Detection and Correction Times	3-43

Section		Page
3.5	Model Formulation with Variable Detection Probability	3-48
3.6	Narrowing the Range Estimates of Model Parameters	3-51
3.7	Variance and Confidence Limits for Model Parameters	3-54
3.8	Comparison of Discrete Models with Weibull Distribution	3-56
4	MODEL IMPLEMENTATION	4-1
4.1	Introduction	4-1
4.2	Capabilities	4-1
4.2.1	Model Versions	4-3
4.2.2	Module Level Models	4-4
4.2.3	System Level Models	4-6
4.3	Model Program Documentation	4-7
5	DATA REQUIREMENTS FOR SOFTWARE RELIABILITY ANALYSIS	5-1
5.1	Requirements	5-1
5.1.1	Test Occasion Date or ID	5-1
5.1.2	Software Errors	5-3
5.1.3	Test Effort	5-6
5.1.4	Weighing Factor	5-7
5.2	Data Assumption	5-7
6	SOFTWARE RELIABILITY REQUIREMENTS SPECIFICATION AND MEASUREMENT	6-1
6.1	Introduction	6-1
6.2	Definition of Reliability	6-2
6.3	Stating Reliability Requirements	6-4
6.4	Measurement of Performance	6-5
6.4.1	Numerical Example	6-6
6.4.2	Confidence Level Considerations	6-8
7	APPLYING THE MODEL TO SOFTWARE DEVELOPMENT PROJECTS	7-1
7.1	Predictions of Future Errors	7-2
7.2	Current and Future Reliability Estimates	7-5
7.3	Example Model Application to Trade-offs	7-10
8	DEFINITION OF TERMS	8-1
9	REFERENCES	9-1



## TABLES

Table	Page
3.2.1-1 Summary of Projects and Errors Analyzed	3-4
3.2.2-1 Data for Project 1	3-10
3.2.2-2 Data for Project 2	3-11
3.2.2-3 Data for Project 3	3-12
3.2.2-4 Data for Projects 4 through 7	3-13
3.2.3-1 Parameter Estimates and Statistics for Project 1	3-14
3.2.3-2 Parameter Estimates and Statistics for Project 2	3-15
3.2.3-3 Parameter Estimates and Statistics for Project 3	3-16
3.2.3-4 Parameter Estimates and Statistics for Project 3, $\alpha = .935$	3-17
3.2.3-5 Parameter Estimates and Statistics for Projects 4 through 7	3-18
3.4-1 Times to Reduce Remaining Errors to a Specified Number	3-45
5.1-1 Historical Data Required for Input to Model	5-2

## FIGURES

Figure	
2.2-1 Error History Characteristics in the Real World of Software Development	2-5
3.2.6.2-1 Simulation Approach	3-30

## FIGURES


Figure		Page
3.2.6.3-1	Simulation Results for Test Condition 2	3-33
3.4-1	Representation of the Error Detection and Correction Process	3-38
5.1.2-1	Relationship Between Problem Reports, Software Changes, and Software Errors	5-4
8-1	Software System Hierarchy	8-3

## EVALUATION

The increased reliance on embedded computers for new weapons systems in such areas as command and control and avionics has led to a resultant increased importance in embedded computer software for these systems. The increasing expenditures on computer software have, as a consequence, led to a desire to produce high quality embedded computer software for low software life-cycle costs. This desire has been expressed in numerous industry and Government sponsored conferences, as well as in documents such as the final reports of the the Joint Commander's Software Reliability Working Group (Nov 1975) and the Joint Logistics Commanders Joint Policy Coordinating Group on Computer Resource Management (Apr 1979). Based upon the recommendations in these documents, numerous efforts have been initiated to develop mathematical techniques for predicting the reliability and error content of a software system for use in management tracking and software qualification testing. However, early efforts have not produced models with the desired predictive accuracy for general model usage.

This effort was initiated in response to this need for developing better and more accurate software error prediction models and fits into the goals of RADC TPO No. 4G, Information Processing (formerly RADC TPO No. 5, Software Cost Reduction), in the subthrust of Software Engineering (Metrics and Modeling). This report summarizes the development of mathematical models based upon the assumption of binomial and Poisson distributions for the times to detect and correct software errors. The importance of this development is that it represents an attempt to include both error generation and non-constant error detection rates into model assumptions, and thus more closely reflects the actual software error detection and correction process.

The theory and equations developed under this effort will lead to needed mathematical techniques for use by software managers in more accurately tracking a software development in terms of reaching prescribed reliability and error content goals. In addition, the more realistic model assumptions will lead to better model predictability, thus insuring more widespread usage of software reliability modeling techniques. Finally, the predictive techniques developed under this effort will be applicable to current Air Force software development projects and thus help to produce the high quality, low cost software needed for today's systems.

  
ALAN N. SUKERT  
Project Engineer

THIS PAGE IS BEST QUALITY PRACTICABLE  
FROM COPY FURNISHED TO DDC

## SECTION 1

### INTRODUCTION

This document is the final technical report for RADC Contract Number F30602-78-C-0346, concerned with the Analysis of Discrete Software Reliability Models. The report is organized topically into nine interrelated sections. A description of the major topics discussed in each section is as follows:

- The remainder of Section 1 presents the purpose of this study and the major results and conclusions.
- Section 2 presents a detailed discussion of the theoretical foundations and mathematical formulation of our discrete reliability models.
- Section 3 presents the results of analysis of these models according to the requirements of our contract with RADC during the period from October 1978 through September 1979. Validation results are presented using historical data provided by RADC from seven development projects. Model validation results using simulated data are also presented. An important topic also presented in this section is the approach used to estimate the time required to a specified number of remaining errors.
- Section 4 briefly elaborates on the APL implementation of these models, particularly when using error data collected at the module or system level.

- Section 5 identifies the data collection requirements for software reliability analysis using the models. A discussion of the rationale and assumptions made about each data item is presented.
- Section 6 contains a discussion of our definition of software reliability and how reliability requirements should be specified and measured.
- Section 7 contains the additional equations needed by management to predict future error occurrences and other statistics related to current status and future reliability predictions. An example of how the models can be applied to trade-off studies during system testing is also provided.
- Sections 8 and 9, respectively, present a definition of important terms (e.g., software error) and technical references used throughout this report.

#### 1.1 BACKGROUND AND PURPOSE

Since 1976 at IBM/FSD in Gaithersburg, Maryland, research activities have been underway to provide models, tools, and procedures for assessing the reliability of a software system at different stages in its development. During this time major attention has been given to the development of discrete software reliability models and their validation using data collected from ongoing projects.

The models as formulated have shown strong indications of their validity when applied to internal projects. Additional mathematical analysis of the models was needed, nevertheless, along with continued validation studies using data from other large-scale projects, to further evaluate

the generalized applicability of the models. The purpose of this contract effort was to perform the required analysis and validation of the models as needed. Section 3 of this report presents what these various analysis requirements were, and the results obtained during validation of the models using historical data provided by RADC and simulated error data developed by IBM.

## 1.2 MAJOR RESULTS AND CONCLUSIONS CONCERNING MODEL VALIDITY

In general the models showed strong indications of their validity when applied both to historical error data, and to simulated error data which was used to more closely examine the effects of random error on the accuracy of the model estimates.

Using historical error data from seven projects, the models showed reasonable fits to the observed software errors and software problem reports. Acceptable estimates of total software errors (N) were obtained in all cases. In those instances where poor fits of the model were obtained the results were viewed as providing an indication of:

- The sensitivity of the models to the assumptions made about the data; and
- The lack of thoroughness of testing of the systems being analyzed.

The results presented in this report point to the need for future data collection efforts to collect error history and test data that meet as closely as possible the model data requirements and assumptions. We consider that the models show a great deal of promise to apply to future software development projects, especially when appropriate data are collected.

The use of simulated error data for purposes of model validation is viewed as a feasible alternative to the use of historical data, until data are collected that more closely agree with the model data requirements and assumptions.

## SECTION 2

### THEORETICAL FOUNDATIONS AND MODEL DEVELOPMENT

#### 2.1 INTRODUCTION AND OVERVIEW

This section discusses the theoretical foundations of the model and presents a thorough discussion of issues related to the mathematical development of the model, methods of parameter estimation, and solution procedure. Prior to this discussion, however, a general overview of the characteristics and capabilities of the model is presented in the context of the work and progress made by IBM/FSD in software reliability research over the last three years.

Since 1976 at IBM/FSD in Gaithersburg, Maryland, important research activities have been underway to provide models, tools, and procedures for predicting the reliability of a software system (see Brooks and Weiler [3] and [4]; Brooks, Kocher, Motley, and Weiler [5]; Motley [13]; and Motley and Brooks [15]). These efforts have been undertaken in recognition of the fact that users of data processing systems, particularly the military, are becoming increasingly concerned with the problems of measuring and predicting the reliability and life cycle costs (which includes continued error detection and correction after delivery) of such systems. Moreover, it was recognized that no tools existed that adequately addressed reliability measurement and prediction in a realistic software development and test environment.

To date, FSD's efforts have resulted in the development of a software reliability model that can be applied at various stages in the development cycle to determine the number of errors remaining, the reliability of the software, and the amount of future test time required to reduce the remaining errors to a specified level.



Research emphasis has been on an incremental approach to model development and refinement. As a result of using data from ongoing software development projects within IBM and data on other large-scale projects supplied to us under contract to the Government, the validity of the model has been examined in a realistic environment. In addition to providing us with validation information, this experience has allowed us to generalize the model to better reflect the realities of software development and testing.

At present, the model takes into account the following realities inherent in the software development and test process:

- The amount of source code under test may vary from one test occasion to the next.
- Each of the software modules of a given system may come in and out of test on various occasions, and are not necessarily under test on all occasions.
- Each module may contribute to the total errors in the system according to its size relative to the total size of the system on any given test occasion (size here could refer to source lines of code, object words of core required, etc.).
- The amount of time spent in testing the system (or a subsystem, module, etc.) may vary from one test occasion to the next.
- Errors may be reinserted in the software during the error removal and correction process; also, the correction of errors exposes additional errors to discovery.

The following represent model outputs which can be used to support management and customer reliability information needs during project development and after delivery:

- The probability that no more than a specified number of software errors of a specified type or classification will occur during some future interval of testing.
- The number of remaining errors (by type, if desired) in a software system prior to delivery, and the amount of future test time required to reduce the errors to a specified level.
- The amount of test time required to get the software system's reliability up to a specified level prior to critical test points (e.g., customer demonstration/acceptance, selloff, delivery) in the development cycle.

Furthermore, the model may be used in trade-off studies that give management clear indications of how much additional test effort should be expended on which system modules to maximize their payoff (e.g., as measured by incentive award dollars/fee given by the customer agency) in meeting the system's reliability requirements test.

## 2.2 PROBLEM STATEMENT

The reliability of any software system, however measured, is an indication of the extent to which the software is suitable or fit to be relied on during actual performance and operation. The extent to which it can be relied on is intimately related to how free from error the system is at present or will be during some future specified period of performance.

The analysis of software errors encountered during previous periods of performance, then, becomes an important activity that would contribute to the measurement and prediction of the software reliability in the future.

For the purposes of this research, the reliability of a software system (subsystem or module) is defined as:

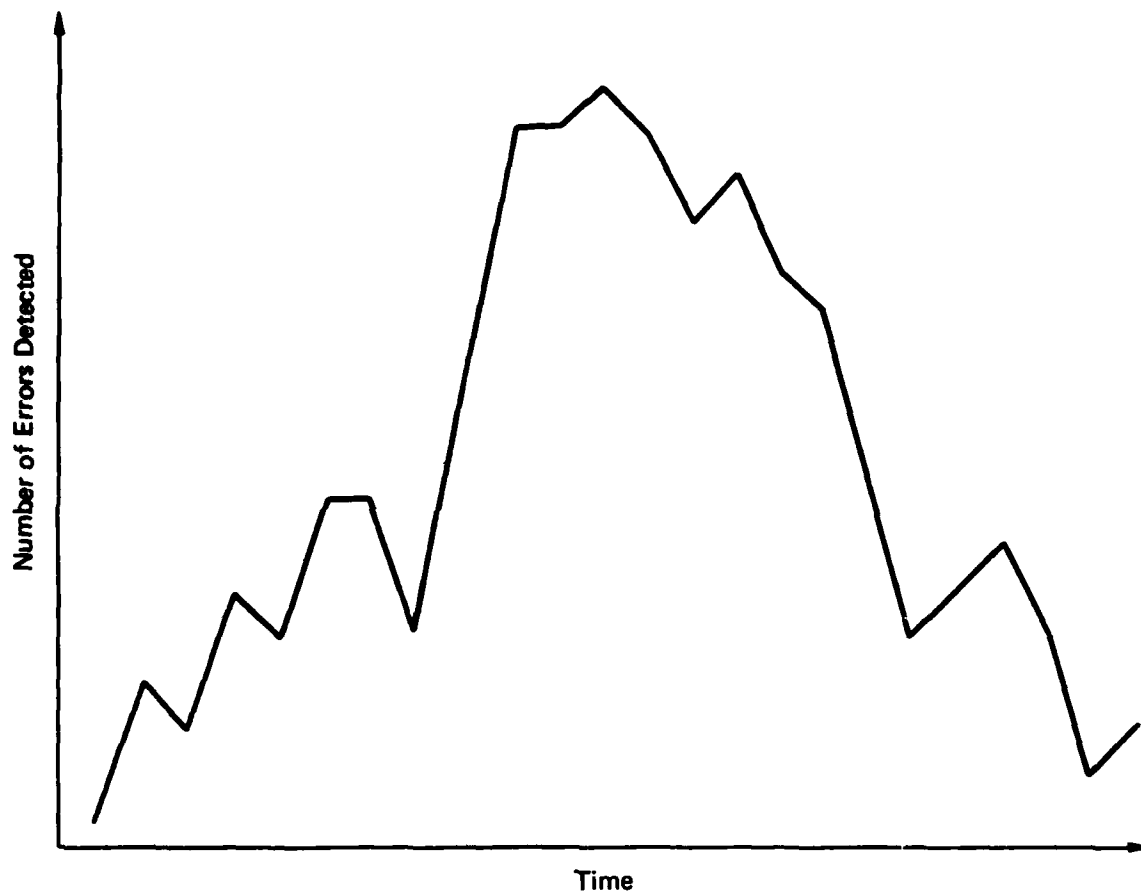
- The probability that no (or no more than a specified number of) software errors of a given type will occur during a specified future time interval under specified testing conditions.

More will be said about this definition in Section 6.0.

Since software reliability is so closely tied to the error-freeness of the system, the analysis of historical errors should necessarily lead to an ability to estimate the finite number of errors that were in the system to begin with, how many are remaining at any given time, and the rate at which errors are being detected and removed. A software reliability model is needed, then, that can estimate these values with increasing precision as the system is developed and tested.

When developing the model to derive these estimates, important factors which influence error occurrences in the real world of software development and testing need to be considered. For example, based on experience in analyzing error data from numerous software development projects, it has been observed that the error histories of these projects generally go through an initial build-up, peaking, and then decaying process similar to that depicted by the hypothetical curve shown in Figure 2.2-1. Admittedly, many factors can be identified which may be contributing to or influence this variability typically observed in project error data. Some of the more important factors assumed to be systematically influencing

Figure 2.2-1. ERROR HISTORY CHARACTERISTICS IN THE REAL WORLD OF SOFTWARE DEVELOPMENT



the number of errors detected and corrected on each occasion are the following:

- The system is being developed and tested incrementally - this means that some portions or modules of the system may be available for testing, or in and out of test, while others are not. This could result from the need for different testing scenarios, phased development and test requirements, or other factors (e.g., lack of personnel for testing).
- During periods of testing, the system or any portion thereof is usually not tested an equal amount from one test occasion to the next.
- Software errors are often reinserted in the programming system in the process of correcting errors already discovered.
- The correction of errors in the system exposes additional errors to detection during future periods of testing.

Given all of this, the problem becomes one of developing a software reliability prediction model which meaningfully relates each of these factors to the occurrence of software errors in the system with the objective of estimating:

- The total number of software errors in the system (N);
- The probability of detecting any one error during a specified unit of testing ( $q$  or  $\phi$ ); and

- The probability of correcting errors in the system without reinserting additional errors and exposing others to discovery ( $\alpha$ ).

Once estimates\* of the model parameters  $N$ ,  $q$  (or  $\phi$ ), and  $\alpha$  are obtained, other important estimates can be derived. These include:

- Errors remaining
- Future errors expected
- Current and future software reliability
- Time required to achieve a specified reliability; and
- Probability of passing a reliability requirements' test.

### 2.3 MATHEMATICAL FORMULATION

The model development approach taken by the FSD researchers has been to consider the process of error detection in software testing as a discrete process and to derive mathematical models to represent this process. Work performed by others in the application of the continuous exponential distribution has been very useful, but leaves much to be desired in terms of precise statement of assumptions, procedures to follow for collection of error data according to the theory postulated by the models, and meaningful conclusions to draw as a result of the analysis made using these models. A key problem results from simply attempting to apply hardware reliability concepts to that of software, leading to, for example, expressions such as "mean-time-to-failure" and "failure rate" for software systems. Many of these concepts become vague, lacking operational meaning and value, when applied to software.

\*Throughout this report no notational distinctions are made between model parameters and their estimates. References to each are readily distinguished by the context in which they are discussed.

Two mathematical models, the binomial and Poisson, have been employed in our model development activity. Each model assumes that:

- The number of software errors detected on each test occasion is proportional to the number of errors at risk for detection; where the number at risk is some portion of the errors remaining.
- The proportionality factor, or probability ( $q$  for the binomial,  $\phi$  for the Poisson) of detecting any one error during a specified unit interval of testing, is constant over all occasions and independent of other error detections.
- The errors reinserted on any occasion are proportional to the number of errors detected.

From the standpoint of parsimony of assumptions, the binomial distribution should be superior to the Poisson. The Poisson is derived from the binomial using the assumption that the population of events (e.g., software errors) is very large and the probability of error detection is very small.\*

Each of these models has two variations. The first variation gives the expected errors in each module of a system under test. The second variation gives the expected errors for the entire portion of the system which is under test. The operational usefulness of these two variations is discussed in Section 4. Their mathematical formulations are presented in the following sections.

---

\*Specifically, the assumption is that  $e^{-q} = 1-q$ .

### 2.3.1 Expected Value for the Binomial Distribution by Module

The expected value for the number of errors detected in module  $j$  in the first unit interval of the first test occasion\* is given by:

$$n'_{1j} = w_j Nq$$

where:

$n'_{1j}$  is the expected number of errors;

$w_j$  is the weight assigned to module  $j$ ;

$N$  is the total number of errors in the system at the beginning of the first test occasion; and

$q$  is the error detection probability, i.e., the probability that any one error will be detected during one unit interval of testing.

For the purpose of our models, the contribution of any module to the number of errors detected on each occasion is considered a function of its size relative to the size of the total system, assuming the module is tested on that occasion. The rationale for this consideration is taken from earlier studies (see Motley and Brooks [14]; Thayer et al. [19]) which strongly indicate that the number of errors in a program is significantly related to the length of the program.

Size here can be measured in terms of source lines of code, object program size (e.g., words of core required to store the object code) or some other relative measure of size. The weight factor,  $w_j$ , is the ratio of the size of the module to that of the total size of the system; i.e., that percentage or portion of the total system size attributable to the  $j$ th module.

\*See Section 8, Definition of Terms



The term  $w_j N$  represents the number of errors exposed to detection; or in classical terms the effective number at risk for module  $j$ .

If all modules are tested on the first occasion  $\sum w_j = 1$ , and all errors are at risk. In the successive unit intervals of the first test occasion we have for module  $j$ :

<u>Unit Interval</u>	<u>Number at Risk</u>	<u>Expected Errors</u>
1	$w_j N$	$w_j Nq$
2	$w_j N(1-q)$	$w_j Nq(1-q)$
3	$w_j N(1-q)^2$	$w_j Nq(1-q)^2$
$\vdots$	$\vdots$	$\vdots$
$t_{1j}$	$w_j N(1-q)^{t_{1j}-1}$	$w_j Nq(1-q)^{t_{1j}-1}$

The total number of errors expected in module  $j$  during the first test occasion is:

$$\begin{aligned}
 n'_{1j} &= \sum_{i=1}^{t_{1j}} w_j N(1-q)^{i-1} q \\
 &= w_j N \left[ 1 - (1-q)^{t_{1j}} \right] \\
 &= w_j N q_{1j}
 \end{aligned}$$

where  $t_{1j}$  is the amount of test effort which is expended on module  $j$  during the first test occasion. In general:

$$q_{ij} = \left[ 1 - (1-q)^{t_{ij}} \right] \quad (2.1)$$

is equal to the effective error detection probability for the  $i$ th test occasion for module  $j$ .

The number of errors at risk in module  $j$  for the second test occasion is:

$$w_j N - n_{1j} + r n_{1j}, \text{ or}$$

$$w_j N - \alpha n_{1j}$$

where:

$n_{1j}$  is the number of software errors actually detected in module  $j$  during the first test occasion;

$r$  is the error reinsertion/uncovery rate which represents two separate effects observed during software development and test. The first effect considers the chance that new errors may have been reinserted in the software during the process of correcting errors already detected. The second effect considers the chance that the correction of errors opens new paths through the software containing errors that were previously not accessible to testing. Both of these effects are assumed to be proportional to the number of errors corrected and are combined into a single factor,  $r$ ; and

$\alpha = 1-r$ ; which is interpreted as the probability of correcting errors in the system without reinserting additional errors and exposing others to discovery. Alpha ( $\alpha$ ) can be estimated by the model or specified as a constant value ( $0 < \alpha \leq 1$ ) when estimating only  $N$  and  $q$  (or  $\phi$ ).

The effective error detection probability  $q_{2j}$  for the second occasion is derived as for the first occasion; thus, the expected number of errors for module  $j$  during the second occasion is:

$$n'_{2j} = (w_j N - \alpha n_{1j}) q_{2j}.$$

For the third occasion it is:

$$n'_{3j} = (w_j N - \alpha n_{1j} - \alpha n_{2j}) q_{3j}$$

For occasion  $i$ :

$$\begin{aligned} n'_{ij} &= (w_j N - \alpha N_{i-1,j}) q_{ij} \\ &= \bar{N}_{ij} q_{ij} \end{aligned} \quad (2.2)$$

where  $N_{i-1,j} = \sum_{m=1}^{i-1} n_{mj}$  is the cumulative number of errors observed for module  $j$  through occasion  $i-1$ ; and

$$\bar{N}_{ij} = w_j N - \alpha N_{i-1,j} \quad (2.3)$$

is the estimate of the number of errors at risk, or remaining in module  $j$ .

### 2.3.2 Expected Value for the Binomial Distribution for the System

The expected number of errors in the portion of the total system which is under test on occasion  $i$  can be found by taking the summation of

(2.3) over all modules which are being tested. The effective number at risk for the system is:

$$\begin{aligned}\bar{N}_i &= \sum_{j \in J_i} (w_j N - \alpha N_{i-1,j}) \\ &= N \sum_{j \in J_i} w_j - \alpha \sum_{j \in J_i} N_{i-1,j}\end{aligned}\quad (2.4)$$

where:

$J_i$  is the set of modules tested on occasion  $i$ ;

$\sum_{j \in J_i}$  denotes that the summation takes place over all modules that are elements of the set  $J_i$ .

$\sum_{j \in J_i} w_j$  is the fraction of the system which is under test; and

$\sum_{j \in J_i} N_{i-1,j}$  is the number of errors detected in that portion of the system prior to the  $i$ th occasion.

It is evident from this formulation that the modules are no longer distinct but are now being considered as a portion of the system. Therefore, it is appropriate to assume that all of the modules being tested during an occasion are under test for the entire occasion. It follows that the test effort for the system may be used in place of the modular test effort. This substitution yields an effective  $q$  for a testing occasion of:

$$q_i = 1 - (1 - q)^{t_i} \quad (2.5)$$

where  $t_i$  is the system test effort on occasion  $i$ .

The expected number of errors in the system on occasion  $i$  is:

$$n_i' = \bar{N}_i q_i \quad (2.6)$$

### 2.3.3 Expected Value for the Poisson Distribution by Module

This distribution is considered by some investigators to be the best one to use for rare events. Also, the Poisson should provide a good approximation to the binomial distribution when  $N$  is large and  $q$  is small. For the Poisson model the error detection rate is assumed to be proportional to the number of errors remaining in each module.

As before, the expected number of errors in module  $j$  at the beginning of test occasion  $i$  is  $\bar{N}_{ij}$ . Therefore, the expected error detection rate for the first unit interval (of length  $t$ ) is  $\bar{N}_{ij} \phi$ , where  $\phi$  is the proportionality factor between errors remaining and error rate. The expected number of errors is given by the product of error rate and unit interval length; i.e.,

$$n_{ij}' = \bar{N}_{ij} \phi t$$

With these definitions in mind the following table is constructed:

<u>Unit Interval</u>	<u>Errors in Module</u>	<u>Expected Error Rate</u>	<u>Expected Errors</u>
1	$\bar{N}_{ij}$	$\bar{N}_{ij} \phi$	$\bar{N}_{ij} \phi t$
2	$\bar{N}_{ij} (1-\phi t)$	$\bar{N}_{ij} (1-\phi t) \phi$	$\bar{N}_{ij} (1-\phi t) \phi t$
$\vdots$	$\vdots$	$\vdots$	$\vdots$
$t_{ij}$	$\bar{N}_{ij} (1-\phi t)^{t_{ij}-1}$	$\bar{N}_{ij} (1-\phi t)^{t_{ij}-1} \phi$	$\bar{N}_{ij} (1-\phi t)^{t_{ij}-1} \phi t$

where:

$t_{ij}$  is the number of unit intervals module  $j$  was tested on occasion  $i$ , and

$t$  is the length of the unit interval (e.g., 1 hour, 1 week, 1 month of testing)

The expected number of errors for module  $j$  on occasion  $i$  is:

$$n'_{ij} = \sum_{m=1}^{t_{ij}} \bar{N}_{ij} \phi t (1-\phi t)^{m-1}$$

With no loss in generality  $t$  may be set equal to unity and then

$$n'_{ij} = \bar{N}_{ij} \phi_{ij} \quad (2.7)$$

where:

$$\phi_{ij} = 1 - (1-\phi)^{t_{ij}}$$

The expected error rate for module  $j$  on occasion  $i$  is:

$$\begin{aligned} \lambda_{ij} &= n'_{ij} / t_{ij} \\ &= \bar{N}_{ij} \phi_{ij} / t_{ij} \end{aligned} \quad (2.8)$$

The equivalence between the above Poisson model and the exponential models of Shooman [17] and Jelinski-Moranda [12] was shown in Brooks and Weiler ([3] and [4]).

#### 2.3.4 Expected Value for the Poisson Distribution for the System

The expected number of errors in the system based on error detection rate is derived using the approach of Section 2.3.2. For occasion  $i$  it is given by:

$$n'_i = \bar{N}_i \phi_i$$

where:

$$\phi_i = 1 - (1-\phi)^{t_i}$$

The expected error rate on occasion i is:

$$\begin{aligned}\lambda_i &= n_i'/t_i \\ &= \bar{N}_i \phi_i / t_i\end{aligned}\tag{2.9}$$

## 2.4 PARAMETER ESTIMATION

Maximum likelihood and least squares methods were used to estimate the parameters for each of the models discussed thus far. Theoretically, each method of estimation when applied should yield reasonably equivalent results. However, in some cases, one method or the other may provide a closer fit to the observed data and more reasonable estimates of N, q (or  $\phi$ ), and  $\alpha$ . When this occurs, the parameters derived from the model and method of estimation providing the closest fit to the observed data should be used. In general, the maximum likelihood approach gives biased estimates, whereas the least squares approach yields unbiased estimates which minimize the variance between observed and expected values.

### 2.4.1 Binomial Maximum Likelihood

By using the binomial distribution and replacing the event probability and number at risk with the effective values as determined in (2.1) and (2.3), the probability of a specified number of errors occurring in module j during test occasion i,  $x_{ij}$ , is:

$$p(x_{ij}) = \binom{\bar{N}_{ij}}{x_{ij}} q_{ij}^{x_{ij}} (1-q_{ij})^{\bar{N}_{ij}-x_{ij}}$$

By replacing  $x_{ij}$  with the actual number of errors observed,  $n_{ij}$ , the likelihood function for a series of observations of  $j$  modules on  $k$  occasions may be written as:

$$L = p(n_{11}, n_{12}, \dots, n_{kj})$$

$$= \prod_{i=1}^k \prod_{j=1}^J \binom{\bar{N}_{ij}}{n_{ij}} q_{ij}^{n_{ij}(1-q_{ij})} \bar{N}_{ij}^{-n_{ij}} \quad (2.10)$$

where:

$k$  is the number of test occasions and

$J$  is the number of modules in the system.

If the factorials\* are replaced with Stirling's formula and the natural logarithm of (2.10) is taken, then the partial derivatives with respect to the unknown parameters  $N$ ,  $q$  and  $\alpha$  may be obtained. When the results are set equal to zero, simultaneous equations are obtained whose solution yields the maximum likelihood estimates of the parameters  $N$ ,  $q$ , and  $\alpha$ .

Noting that terms of the form:

$$\frac{c}{2\bar{N}_{ij}} - \frac{c}{2(\bar{N}_{ij} - n_{ij})} \quad (2.11)$$

(where  $c$  is a constant) will be small enough to be neglected in most cases, the partial derivatives when set equal to zero are:

$$\frac{\partial \ln L}{\partial N} \equiv 0 = \sum_{i=1}^k \sum_{j=1}^J \left[ w_j \ln \frac{\bar{N}_{ij}}{\bar{N}_{ij} - n_{ij}} + w_j t_{ij} \ln(1-q) \right] \quad (2.12)$$

\*The term  $\binom{\bar{N}_{ij}}{n_{ij}}$  is equivalently written in factorial notation as:

$$\frac{\bar{N}_{ij}!}{n_{ij}! (\bar{N}_{ij} - n_{ij})!}.$$



$$\frac{\partial \ln L}{\partial q} \equiv 0 = \sum_{i=1}^k \sum_{j=1}^J \left[ \frac{n_{ij} t_{ij}}{1 - (1-q)^{t_{ij}}} - t_{ij} \bar{N}_{ij} \right] \quad (2.13)$$

$$\frac{\partial \ln L}{\partial \alpha} \equiv 0 = \sum_{i=1}^k \sum_{j=1}^J \left[ N_{i-1,j} \ln \frac{\bar{N}_{ij}}{\bar{N}_{ij} - n_{ij}} + N_{i-1,j} t_{ij} \ln(1-q) \right] \quad (2.14)$$

Equations (2.12), (2.13), and (2.14) represent simultaneous equations that must be solved to find the maximum likelihood estimates of  $N$ ,  $q$ , and  $\alpha$  for the binomial module level model.

It is not unusual that some module  $j$  would not be tested at all on some test occasion  $i$ . In that case the respective values for  $t_{ij}$  and  $n_{ij}$  should be set to zero. Application of l'Hopital's rule shows that the first term in (2.13) is indeterminate under these conditions. It should then also be set equal to zero.

Inspection of (2.12) and (2.14) reveals that the denominator in the first logarithmic term is the effective number of errors at risk minus the observed number of errors. For some sets of observations where the data is erratic, the introduction of weighting factors for individual modules causes the effective errors at risk to be less than the observed errors, especially if the weight estimate is too small. The argument of the logarithm then becomes negative which precludes convergence. This anomaly may be avoided by selecting the system level variation of the model as given by (2.6).

For the binomial system level model, the likelihood function now becomes:

$$L = p(n_1, n_2, n_3, \dots, n_k) \\ = \prod_{i=1}^k \binom{\bar{N}_i}{n_i} q_i^{n_i} (1-q_i)^{\bar{N}_i - n_i}$$

where:

$$n_i = \sum_{j=1}^J n_{ij}$$

Proceeding as before the partial derivatives when set equal to 0 are:

$$\frac{\partial \ln L}{\partial N} = \sum_{i=1}^k \ln \left[ \frac{\bar{N}_i}{\bar{N}_i - n_i} \right] + t_i \ln(1-q) \sum_{j \in J_i} w_j = 0$$

$$\frac{\partial \ln L}{\partial q} = \sum_{i=1}^k \left[ \frac{n_i t_i}{1 - (1-q) t_i} + t_i \bar{N}_i \right] = 0$$

$$\frac{\partial \ln L}{\partial \alpha} = \sum_{i=1}^k \left[ \ln \frac{\bar{N}_i}{\bar{N}_i - n_i} + t_i \ln(q) \right] \sum_{j \in J_i} N_{i-1,j} = 0$$

#### 2.4.2 Poisson Maximum Likelihood

For the Poisson distribution at the module level, the density function for number of errors  $n_{ij}$  with parameter  $\mu_{ij}$  is:

$$f(n_{ij}, \mu_{ij}) = \frac{\mu_{ij}^{n_{ij}} e^{-\mu_{ij}}}{n_{ij}!},$$

Setting  $\mu_{ij} = \lambda_{ij} t_{ij}$ , and using the expression for error rate  $\lambda_{ij}$  as given by (2.8), we get:

$$f(n_{ij}, \bar{N}_{ij} \phi_{ij}) = \frac{(\bar{N}_{ij} \phi_{ij})^{n_{ij}} e^{-\bar{N}_{ij} \phi_{ij}}}{n_{ij}!}$$

The likelihood function is:

$$L = \prod_{i=1}^k \prod_{j=1}^J \frac{(\bar{N}_{ij} \phi_{ij})^{n_{ij}} e^{-\bar{N}_{ij} \phi_{ij}}}{n_{ij}!}$$

Proceeding in the same manner as in the binomial case the natural logarithm of the likelihood is taken and the partial derivatives with respect to  $N$ ,  $\alpha$  and (in this case)  $\phi$  are developed and set equal to zero:

$$\begin{aligned} \frac{\partial \ln L}{\partial N} &\equiv 0 = \sum_{i=1}^k \sum_{j=1}^J w_j \left[ \frac{n_{ij}}{\bar{N}_{ij}} - \phi_{ij} \right] \\ \frac{\partial \ln L}{\partial \phi} &\equiv 0 = \sum_{i=1}^k \sum_{j=1}^J t_{ij} (1-\phi)^{t_{ij}} \left[ \frac{n_{ij}}{1-(1-\phi)^{t_{ij}}} - \bar{N}_{ij} \right] \\ \frac{\partial \ln L}{\partial \alpha} &\equiv 0 = \sum_{i=1}^k \sum_{j=1}^J N_{i-1,j} \left[ \frac{n_{ij}}{\bar{N}_{ij}} - \phi_{ij} \right] \end{aligned}$$

At the system level the Poisson distribution is:

$$f(n_i, \lambda_i, t_i) = \frac{(\lambda_i t_i)^{n_i} e^{-\lambda_i t_i}}{n_i!}$$

Using the expression for  $\lambda_i$  as given by (2.9), the likelihood function is:

$$L = \prod_{i=1}^k \frac{(\bar{N}_i \phi_i)^{n_i} e^{-\bar{N}_i \phi_i}}{n_i!}$$

After taking the logarithm of the likelihood, the partial derivatives of  $N$ ,  $\phi$  and  $\alpha$  are developed and equated to zero, which yields:

$$\begin{aligned}\frac{\partial \ln L}{\partial N} &\equiv 0 = \sum_{i=1}^k \left( \sum_{j \in J_i} w_j \right) \left( \frac{n_i}{\bar{N}_i} - \phi_i \right) \\ \frac{\partial \ln L}{\partial q} &\equiv 0 = \sum_{i=1}^k t_i (1-\phi)^{t_i} \left[ \frac{n_i}{1-(1-\phi)^{t_i}} - \bar{N}_i \right] \\ \frac{\partial \ln L}{\partial \alpha} &\equiv 0 = \sum_{i=1}^k \left( \sum_{j \in J_i} N_{i-1,j} \right) \left( \frac{n_i}{\bar{N}_i} - \phi_i \right)\end{aligned}$$

#### 2.4.3 Least Squares

Two least squares approaches were applied to the parameter estimation problem. In the first approach (module level), the individual differences between the observed and predicted errors for each module on each test occasion were used.

In the second approach (system level), the errors are summed before comparison so that all of the errors which are estimated for a test occasion are compared with the total observed errors for that occasion.

The least squares approach using the expected value equations for the binomial or Poisson distribution yields formally identical partial derivatives with respect to  $N$ ,  $q$  (or  $\phi$ ), and  $\alpha$ . When these are solved, identical estimates of the parameters are obtained, regardless of whether the expected value equation for the binomial or Poisson is used. For this reason the least square model version is only distinguished with respect to the level (i.e., module or system) at which it is applied. The model equations in the following two sections are derived using the expected value equations for the binomial distribution.

#### 2.4.3.1 Least Squares by Module

The expected value of the number of errors detected in module  $j$  during test occasion  $i$  is given by (2.2). The actual number of observed errors is  $n_{ij}$ . The parameter estimates  $N$ ,  $q$  and  $\alpha$  are to be determined such that the sum of the squared differences between the observed values and the expected values is minimized. The squared residual function is given by:

$$\begin{aligned} R &= \sum_{i=1}^k \sum_{j=1}^J (n_{ij} - \bar{n}_{ij})^2 \\ &= \sum_{i=1}^k \sum_{j=1}^J (n_{ij} - \bar{N}_{ij} q_{ij})^2 \end{aligned}$$

The partial derivatives of  $R$  with respect to the parameters  $N$ ,  $q$  and  $\alpha$ , when set equal to zero, are:

$$\begin{aligned} \frac{\partial R}{\partial N} &\equiv 0 = \sum_{i=1}^k \sum_{j=1}^J \left[ (n_{ij} - \bar{N}_{ij} q_{ij}) w_j q_{ij} \right] \\ \frac{\partial R}{\partial q} &\equiv 0 = \sum_{i=1}^k \sum_{j=1}^J \left[ (n_{ij} - \bar{N}_{ij} q_{ij}) \bar{N}_{ij} t_{ij} (1 - q_{ij}) \right] \\ \frac{\partial R}{\partial \alpha} &\equiv 0 = \sum_{i=1}^k \sum_{j=1}^J \left[ (n_{ij} - \bar{N}_{ij} q_{ij}) N_{i-1,j} q_{ij} \right] \end{aligned}$$

#### 2.4.3.2 Least Squares at the System Level

The expected value for the number of errors detected in the system during test occasion  $i$  is given by (2.6). The observed number of errors is  $n_i$ .

The squared residual function is given by:

$$R = \sum_{i=1}^k \left[ n_i - \bar{N}_i q_i \right]^2$$

The partial derivatives of R with respect to N, q and  $\alpha$ , when set equal to zero, are:

$$\frac{\partial R}{\partial N} \equiv 0 = \sum_{i=1}^k \left[ \left( n_i - \bar{N}_i q_i \right) q_i \sum_{j \in J_i} w_j \right]$$

$$\frac{\partial R}{\partial q} \equiv 0 = \sum_i^k \left[ \left( n_i - \bar{N}_i q_i \right) \bar{N}_i t_i (1 - q_i) \right]$$

$$\frac{\partial R}{\partial \alpha} \equiv 0 = \sum_{i=1}^k \left[ \left( n_i - \bar{N}_i q_i \right) q_i \sum_{j \in J_i} N_{i-1,j} \right]$$

## 2.5 SOLUTION PROCEDURES

Each of the parameter estimation techniques in the previous section resulted in three simultaneous non-linear equations involving the three unknowns N,  $\alpha$  and q ( $\phi$  for the Poisson case). These equations were solved by using the Newton-Raphson method (see Hildebrand [10] and Dodes [6]) generalized for the treatment of three equations. In a more compact notation the equations may be written:

$$F(N, q, \alpha) = 0$$

$$G(N, q, \alpha) = 0$$

$$H(N, q, \alpha) = 0$$

where the functions  $F$ ,  $G$ , and  $H$  represent the partial derivatives of the estimating function with respect to  $N$ ,  $q$  (or  $\phi$ ), and  $\alpha$ , respectively.

In preparation for an iterative technique, the following linear simultaneous equations are required:

$$\Delta N F_N + \Delta q F_q + \Delta \alpha F_\alpha = -F(N_i, q_i, \alpha_i)$$

$$\Delta N G_N + \Delta q G_q + \Delta \alpha G_\alpha = -G(N_i, q_i, \alpha_i)$$

$$\Delta N H_N + \Delta q H_q + \Delta \alpha H_\alpha = -H(N_i, q_i, \alpha_i)$$

where:

$$F(N_i, q_i, \alpha_i)$$

$$G(N_i, q_i, \alpha_i)$$

$$H(N_i, q_i, \alpha_i)$$

denotes that these respective partial derivatives are evaluated for estimated values of  $N$ ,  $q$ , and  $\alpha$  at the  $i$ th iteration; ( $F_N$ ,  $F_q$ ,  $F_\alpha$ ) are defined as:

$$F_N = \frac{\partial F(N_i, q_i, \alpha_i)}{\partial N}$$

$$F_q = \frac{\partial F(N_i, q_i, \alpha_i)}{\partial q}$$

$$F_\alpha = \frac{\partial F(N_i, q_i, \alpha_i)}{\partial \alpha};$$

( $G_N$ ,  $G_q$ ,  $G_\alpha$ ) and ( $H_N$ ,  $H_q$ ,  $H_\alpha$ ) are similarly defined as the second-order partial derivatives of the functions  $G$  and  $H$  each with respect to ( $N$ ,  $q$ , and  $\alpha$ ) evaluated at the point ( $N_i$ ,  $q_i$ ,  $\alpha_i$ ); and

$$[\Delta N, \Delta q, \Delta \alpha]$$

are the unknown correction factors that must be solved for at each iteration. Once obtained, these values are then substituted in the equations:

$$N_{i+1} = N_i + \Delta N$$

$$q_{i+1} = q_i + \Delta q$$

$$\alpha_{i+1} = \alpha_i + \Delta \alpha$$

giving us a new estimate of  $N$ ,  $q$ , and  $\alpha$  for the next iteration.

This Newton-Raphson procedure has been implemented in APL at IBM in Gaithersburg, Maryland. In this environment the iterative procedure is continued until the values of  $F$ ,  $G$ , and  $H$  are  $10^{-6}$  or less. More will be said about this APL implementation of the models in Section 4.

## 2.6 CORRECTNESS VERIFICATION

The correctness of the parameter estimation techniques is checked by developing several sets of exact observations of  $n_i$ ,  $t_i$  and  $w_i$  using the basic assumptions of the model and some selected values of  $N$ ,  $q$ , and  $\alpha$ . These exact observations are then used independently by the solution procedures to determine estimates for the parameters. If the estimated parameters agree exactly with the values which were initially selected then the derivation of the models and parameter estimation techniques are considered to be correct. It is of interest to note that the neglect of the terms in (2.11) exactly compensate for the approximation introduced by Stirling's formula, with the result that the exact agreement as noted above is obtained. The correctness of the implementation of the solution procedure has also been verified by introducing slight perturbations into the exact observations as previously developed. If the solution procedures yield estimates which are close to the selected values then the convergence procedures are also assumed to be correct.



Correctness checks were successfully performed on all model equations that appear in this section.

## 2.7 INITIAL VALUE AND CONVERGENCE CONSIDERATIONS

The Newton-Raphson approach was found to be quite accurate for the purposes of our research. However, it requires initial estimates of  $N$ ,  $q$  and  $\alpha$  which must either be close to the actual values or else related to each other in some manner that is not readily apparent. When using this procedure some solutions were readily found, while in other problems many different sets of estimates were tried but no solution was ever found. Occasionally the procedure would converge to a "local maximum", where the parameters did not appear to be consistent with the observed data. In these cases additional estimates were tried until a reasonable set of solutions was obtained.

Furthermore, in experimental cases where the three parameters were to be estimated and the test effort data ( $t_i$ ) was constant from one occasion to the next, convergence to a solution was either not obtained, or convergence to values that were only close to the known solution occurred. If, however, only two of the parameters ( $N$  and  $q$ ) were estimated, with  $\alpha$  being held constant at some specified value, then convergence to the known values of  $N$  and  $q$  was readily attained. This behavior essentially indicates that the three equations resulting from the partial differentiation are not independent when the values for test effort ( $t_i$ ) are equal over all occasions.

These issues and considerations have directly led us to further examine the range estimates of the model parameters. The results of that contract task are discussed in Section 3.6.

SECTION 3  
MODEL ANALYSIS REQUIREMENTS AND RESULTS

3.1 INTRODUCTION

This section presents the results of various model analysis tasks that were performed under contract with RADC during the period from October 1978 thru September 1979. A brief description of the topics discussed in each subsection is as follows:

<u>Section</u>	<u>Description</u>
3.2	Presents a discussion of model validation results and conclusions based on historical error data provided by RADC. Results and conclusions obtained using simulated data are also provided.
3.3	Discusses the results obtained by simulation to determine an optimum time interval for aggregation of data.
3.4	Presents a mathematical approach for estimating the total error detection and correction time required to reduce the remaining errors to a specified number.
3.5	Discusses a model formulation which allows for a variable probability of error detection.

<u>Section</u>	<u>Description</u>
3.6	Discusses results and recommendations for narrowing the range estimates of $N$ , $q$ (or $\phi$ ), and $\alpha$ when the test effort is constant over all occasions.
3.7	Discusses findings with respect to determining the variance estimates and confidence limits for the model parameters.
3.8	Discusses the application of the Weibull distribution to the analysis of error history data.

### 3.2 MODEL VALIDATION

This section presents and discusses the model validation results obtained using the binomial, Poisson, and least squares models described in Section 2. Seven sets of historical error data were supplied by RADC for this purpose. A brief description of this data follows.

#### 3.2.1 Historical Error Data Analyzed

Table 3.2.1-1 provides a summary of the project characteristics from which each set of data was taken. Essentially all seven projects were each individually involved in the development of a large-scale, command and control software system. Each had used some version of the JOVIAL\* language during development, with the exception of Project 1 which used CENTRAN\* and Assembly Language (ALC). For each project, the error data had been collected during various stages of formal system test and integration. For Projects 4 through 7 the data also included errors observed during the actual customer acceptance test and during the initial period of system operation.

The error data effectively represents a history of the software problem reports (SPRs) recorded during formal testing. As discussed earlier in Section 2, the models have been developed to analyze valid software errors. For our purposes, a software error is defined as:

- Software error - any one of a number of errors that can be classified as being caused by or attributed to the activity of computer programming, and that can be corrected by a change to the software itself.

Software problem reports according to our definition (see Section 8, Definition of Terms) represent symptoms of latent software errors in the

---

\*Both JOVIAL and CENTRAN are higher order languages.

Table 3.2.1-1. SUMMARY OF PROJECTS AND ERRORS ANALYZED (Page 1 of 2)

<u>Project</u>	<u>System</u>	<u>Reference</u>	<u>System Size</u>	<u>Language</u>	<u>Test Phase</u>	<u>Time Period</u>
1	Command, Control, & Communications	Baker [2]	1317K	ALC, CENTRAN	T&I (a)	3/74-2/75
2	Defense, Ground Based Radar (Build F only)	Willman et al. [20]	124K	Jovial, J3	T&I	1/73-11/75
3	Defense, Avionics/Radar	Fries [7]	80K 40K	ALC Jovial, J3B	T&I	8/74-8/75
4 - 7	Command & Control	Thayer et al. [19]	115K	Jovial, J4	T&I Acceptance, Operation	7/73-11/73

(a) Refers to formal test and integration phase.

Table 3.2.1-1. SUMMARY OF PROJECTS AND ERRORS ANALYZED (Page 2 of 2)

<u>Project</u>	<u>Problem Reports</u>	<u>Valid Errors</u>	<u>No. of Occasions</u>
1	5937	2657	12 Months
2	1399	1301	35 Months
3	1672	1239	13 Months
4(a)	Not available	1138	15 Weeks
5	Not available	1483	15 Weeks
6	Not available	2707	15 Weeks
7	Not available	2362	15 Weeks

(a) Projects 4, 5, 6, and 7 constitute one set of error data counted four different ways. The rationale for these counts is reported in Sukert [18] and is based on attempting to remove different types of non-software related errors from the total set of SPRs reported during testing.

system. Until a problem report can be reliably classified as being caused by the software its value in actual model applications becomes questionable. In order to arrive at a more reliable measure of valid software errors, various error categories were eliminated from the total number of problem reports recorded against each project's software.

For Projects 1, 2, and 3 the following categories of errors for each test occasion were eliminated from the analysis:

<u>Problem Report Error Classification</u>	<u>Description</u>
ØØ	All data records having blank error codes
EE	Operating system errors
LL	User requested changes
PP	Recurrent errors (i.e., duplicate reports)
QQ	Documentation errors
SS	Unidentified errors (i.e., errors not able to be classified)
TT	Operator errors
UU	Questions
VV	Hardware errors
WW	Design/requirements logic error

The error categories for these three projects had been classified according to the scheme reported by Thayer et al. [19], thus making the error elimination process quite straightforward.

The data for Projects 4 through 7 were used in their entirety as reported in Sukert [18]. In his preparation of the data for model analysis, a similar, although less extensive, elimination of non-software related problem reports was also accomplished.

In addition to our analyses of the data, several researchers have used either the same SPR data or some variant of it in the analysis and validation of other software reliability models. For other analyses of data

from Projects 1, 2, and 3, see Schafer et al. [16]; for Projects 4 through 7, see reports by Sukert [18] and Goel [9]. Variants of the error counts for each project's data result for several reasons. The most likely reason is that different authors are likely to count the number of SPRs and software errors to be used in the analysis in different ways. What may appear as a valid SPR or software error for one researcher may be invalid and not appropriate to be counted by another. In effect each author has his own definition of what constitutes a valid software error or problem.

To facilitate comparison of the model results with others using like sets of data, the SPRs from Projects 1, 2, and 3 have been included in the analysis. The SPRs are not part of the analysis for the remaining projects (4 to 7), since that data was taken directly from Sukert's [18] published report containing the filtered software errors. Another reason for including SPRs is that the SPR data may be the only available data to work from. Our experience has indicated that, under certain test conditions from one occasion to the next, a consistent proportion of valid software errors will often result for a given number of observed problem reports. When this is observed, the number of valid software errors on each occasion may be predictable or proportionately estimated from an analysis of the SPRs. In other words, a theory which accounts for one will also account for the other.

Readers interested in a more detailed discussion of the projects from which the error data has been collected should consult the appropriate reports referenced in Table 3.2.1-1.

### 3.2.2 Data Assumptions

Various assumptions had to be made about the data prior to its analysis. In general, for all projects it was assumed that the entire system was completely under test (i.e.,  $w_i$  for the system was set to 1.0) over all



occasions. This assumption was made because no data were available concerning the actual portions of the system under test. This assumption is a critical one, particularly in the early stages of testing. The impact of this assumption is that the errors in the system are underestimated.

A second assumption made for all projects was that no errors were reinserted in the software during the process of correcting errors already detected (i.e., an assumption of  $\alpha = 1$ ). To assume that  $\alpha$  is less than one using historical data would imply that additional data were available to aid in establishing the validity of  $\alpha$  estimated by the model. No such data was available or reported. In the case of Project 2, however, a conservative estimate of the error reinsertion rate was put at 6.5% (Fries [7, p. 18]), hence an  $\alpha$  of 0.935. For this project then, N and q (or  $\phi$ ) were estimated once for each assumption  $\alpha = 1.0$  and  $\alpha = .935$  in the analysis for software errors.

A final assumption had to be made about the test effort data for Project 3. For this project, the test time for each occasion was based on the date the first SPR was written against the software function tested on that occasion. According to the assumptions made for calculating the test effort as reported in Fries [7, pp. 19-20], the resultant measure of testing was highly questionable because of extremely high variance, and therefore was not used.

As an alternative, an estimate of system test days was computed according to the assumption that the system was under test only on those days when errors were reported against it. Under this assumption, the measure of testing effort will tend to be underestimated on those occasions when the system was actually tested but no errors were found. As time goes on this effect is likely to become more prominent. Underestimating test effort will lead to overestimating errors (i.e., N), since the time per error will be too low implying that errors are easier to find than is really the case.

The error and test effort data for each of the projects is reported in Tables 3.2.2-1 through 3.2.2-4.

For comparison purposes, all SPRs and errors were also analyzed assuming that the test effort was constant over all occasions. This was done to allow readers to compare the results with other models (which made this assumption), as well as to compare each of our models under the two assumptions. If the results are always about the same, the adjustment for unequal times is unnecessary. If they are significantly different, then it becomes necessary to collect more exact data on future projects.

### 3.2.3 Results

Model analysis results for each project are presented in the following tables as indicated below:

<u>Project</u>	<u>Table</u>
1	3.2.3-1
2	3.2.3-2
3 ( $\alpha=1$ )	3.2.3-3
3 ( $\alpha=.935$ )	3.2.3-4
4,5,6,7	3.2.3-5

Table 3.2.2-1. DATA FOR PROJECT 1

		-----Error Data-----		-----Test Effort Data-----	
		Software	Valid Software	(t <sub>i</sub> )	
Occasion		Problem Reports	Errors	Variable (a)	Constant
<u>1974</u>	3	726	411	1215.94	1
	4	860	411	931.94	1
	5	695	362	849.67	1
	6	814	192	960.23	1
	7	505	228	759.92	1
	8	503	229	854.87	1
	9	374	175	1086.87	1
	10	390	201	829.34	1
	11	356	177	934.01	1
	12	234	104	730.08	1
<u>1975</u>	1	277	109	722.93	1
	2	203	58	571.25	1
Total:		5937	2657		

(a) The number of CPU hours the system was under test on each occasion.

Table 3.2.2-2. DATA FOR PROJECT 2

Occasion	-----Error Data-----		-----Test Effort Data-----	
	Software Problem Reports	Valid Software Errors	Variable <sup>(a)</sup>	(t <sub>i</sub> ) Constant
<u>1973</u>				
1	8	7	7.25	1
2	22	22	3.17	1
3	33	32	7.08	1
4	51	47	7.33	1
5	28	26	7.25	1
6	26	25	12.58	1
7	16	16	19.92	1
8	48	48	52.50	1
9	39	36	47.18	1
10	57	53	95.10	1
11	63	57	55.75	1
12	43	39	59.25	1
<u>1974</u>				
1	77	71	43.58	1
2	83	80	44.75	1
3	68	65	42.33	1
4	65	57	75.00	1
5	96	90	62.83	1
6	62	60	73.58	1
7	65	57	42.75	1
8	97	90	40.67	1
9	51	46	96.75	1
10	64	57	88.58	1
11	36	29	56.75	1
12	42	40	79.25	1
<u>1975</u>				
1	18	16	73.50	1
2	21	18	65.33	1
3	39	37	67.83	1
4	15	15	116.92	1
5	8	8	88.08	1
6	28	28	78.08	1
7	7	6	37.92	1
8	5	5	41.08	1
9	3	3	54.50	1
10	3	3	63.00	1
11	12	12	39.50	1
Total:	1399	1301		

(a) Number of wall clock hours the system was under test on each occasion.

Table 3.2.2-3. DATA FOR PROJECT 3

-----Error Data-----			-----Test Effort Data-----	
Occasion	Software		Variable <sup>(a)</sup>	(t <sub>i</sub> ) Constant
	Problem Reports	Valid Software Errors		
<u>1974</u> 8	62	44	13	1
9	183	135	23	1
10	279	210	25	1
11	199	162	21	1
12	149	97	16	1
<u>1975</u> 1	63	39	25	1
2	96	72	20	1
3	107	83	22	1
4	201	147	23	1
5	181	127	24	1
6	101	77	19	1
7	46	41	13	1
8	5	5	3	1
Total:	1672	1239		

(a) An estimate of variable test effort was used here; it was computed according to the procedure described in Section 3.2.2.

Table 3.2.2-4. DATA FOR PROJECTS 4 THROUGH 7<sup>(a)</sup>

<u>Occasion</u>	<u>Valid Software Errors</u>			
	<u>Project 4</u>	<u>Project 5</u>	<u>Project 6</u>	<u>Project 7</u>
1	203	238	369	334
2	136	179	298	255
3	183	222	398	359
4	47	73	115	89
5	46	56	83	73
6	71	88	150	133
7	54	78	142	118
8	57	92	172	137
9	80	104	202	178
10	64	85	168	147
11	27	53	89	63
12	42	45	87	84
13	55	59	111	107
14	62	84	253	231
15	11	27	70	54
Total:	1138	1483	2707	2362

(a) Testing effort was assumed constant ( $t_i = 1$ ) over all occasions.  
Software errors were counted on a weekly basis for these projects.

Table 3.2.3-1. PARAMETER ESTIMATES AND STATISTICS FOR PROJECT 1

<u>Variable <math>t_i</math></u>	<u>N</u>	<u>q(or <math>\phi</math>)</u>	<u>r</u>	<u><math>r^2</math></u>	<u><math>\chi^2</math></u>
<u>Software Errors</u>					
Binomial	3771	$1.17 \times 10^{-4}$	.89	.79	112
Poisson	3721	$1.20 \times 10^{-4}$	.90	.81	112
Least Squares	3902	$1.08 \times 10^{-4}$	.89	.79	117
<u>Constant <math>t_i</math></u>					
Binomial	3243	.1329	.95	.90	67
Poisson	3228	.1341	.95	.90	67
Least Squares	3254	.1320	.95	.90	67
Observed Errors:	2657				
No. of Occasions:	12				
<u>Problem Reports</u>					
<u>Variable <math>t_i</math></u>					
Binomial	9337	$9.67 \times 10^{-5}$	.84	.71	247
Poisson	9180	$9.96 \times 10^{-5}$	.85	.72	246
Least Squares	(a)				
<u>Constant <math>t_i</math></u>					
Binomial	7655	.1171	.94	.88	114
Poisson	7632	.1177	.94	.88	113
Least Squares	7857	.1117	.94	.88	117
No. of SPRs:	5937				
No. of Occasions:	12				

(a) No convergence obtained.

Table 3.2.3-2. PARAMETER ESTIMATES AND STATISTICS FOR PROJECT 2

<u>Variable <math>t_i</math></u>	<u>N</u>	<u>q(or <math>\phi</math>)</u>	<u>r</u>	<u><math>r^2</math></u>	<u><math>\chi^2</math></u>
<u>Software Errors</u>					
Binomial	1438	$1.27 \times 10^{-3}$	.57	.32	567
Poisson	1423	$1.32 \times 10^{-3}$	.57	.32	559
Least Squares	1617	$8.32 \times 10^{-4}$	.53	.28	761
<u>Constant <math>t_i</math></u>					
Binomial	2500	.0208	.28	.08	530
Poisson	2338	.0229	.28	.08	532
Least Squares	3225	0.148	.28	.08	530
Observed Errors:	1301				
No. of Occasions:	35				
<u>Problem Reports</u>					
<u>Variable <math>t_i</math></u>					
Binomial	1549	$1.26 \times 10^{-3}$	.57	.33	601
Poisson	1533	$1.31 \times 10^{-3}$	.57	.33	593
Least Squares	1749	$8.22 \times 10^{-4}$	.54	.29	805
<u>Constant <math>t_i</math></u>					
Binomial	2729	.0203	.27	.07	577
Poisson	2545	.0224	.27	.07	580
Least Squares	3545	.0144	.27	.07	577
No. of SPRs:	1399				
No. of Occasions:	35				



Table 3.2.3-3. PARAMETER ESTIMATES AND STATISTICS FOR PROJECT 3

<u>Variable <math>t_i</math></u>	<u>N</u>	<u>q(or <math>\phi</math>)</u>	<u>r</u>	<u><math>r^2</math></u>	<u><math>\chi^2</math></u>
<u>Software Errors</u>					
Binomial	3094	$2.07 \times 10^{-3}$	.75	.56	164
Poisson	2801	$2.35 \times 10^{-3}$	.75	.56	163
Least Squares	2660	$2.57 \times 10^{-3}$	.75	.56	160
<u>Constant <math>t_i</math></u>					
Binomial	2075	.0676	.46	.21	322
Poisson	1900	.0771	.46	.21	323
Least Squares	2156	.0639	.46	.21	323
Observed Errors:	1239				
No. of Occasions:	13				
<u>Problem Reports</u>					
<u>Variable <math>t_i</math></u>					
Binomial	4062	$2.15 \times 10^{-3}$	.76	.58	213
Poisson	3696	$2.42 \times 10^{-3}$	.76	.58	213
Least Squares	3708	$2.46 \times 10^{-3}$	.76	.58	209
<u>Constant <math>t_i</math></u>					
Binomial	2768	.0688	.47	.22	430
Poisson	2532	.0787	.47	.22	434
Least Squares	2911	.0639	.47	.22	430
No. of SPRs:	1672				
No. of Occasions:	13				

Table 3.2.3-4. PARAMETER ESTIMATES AND STATISTICS FOR PROJECT 3,  $\alpha=.935$

<u>Variable <math>t_i</math></u>	<u>N</u>	<u>q(or <math>\phi</math>)</u>	<u>r</u>	<u><math>r^2</math></u>	<u><math>\chi^2</math></u>
<u>Software Errors</u>					
Binomial	2918	$2.19 \times 10^{-3}$	.75	.56	164
Poisson	2620	$2.52 \times 10^{-3}$	.75	.56	163
Least Squares	2488	$2.75 \times 10^{-3}$	.75	.56	161
<u>Constant <math>t_i</math></u>					
Binomial	1954	.0715	.46	.21	322
Poisson	1776	.0825	.46	.21	323
Least Squares	2016	.0683	.46	.21	323
Observed Errors:	1239				
No. of Occasions:	13				

Table 3.2.3-5. PARAMETER ESTIMATES AND STATISTICS FOR PROJECTS 4 THROUGH 7

	<u>N</u>	<u>q(or <math>\phi</math>)</u>	<u>r</u>	<u>r<sup>2</sup></u>	<u><math>\chi^2</math></u>
<u>Project 4</u>					
Binomial	1348	.1166	.83	.69	164
Poisson	1324	.1210	.83	.69	165
Least Squares	1241	.1395	.83	.69	186
<u>Project 5</u>					
Binomial	1823	.1060	.84	.70	155
Poisson	1798	.1088	.84	.70	155
Least Squares	1693	.1230	.84	.70	170
<u>Project 6</u>					
Binomial	3958	.0739	.68	.46	477
Poisson	3793	.0793	.68	.46	480
Least Squares	3486	.0916	.68	.46	504
<u>Project 7</u>					
Binomial	3446	.0742	.66	.43	487
Poisson	3278	.0806	.66	.43	491
Least Squares	2989	.0946	.66	.43	520
Project:	<u>4</u>	<u>5</u>	<u>6</u>	<u>7</u>	
Observed Errors:	1138	1483	2707	2362	
No. of Occasions:	15	15	15	15	

### 3.2.3.1 Goodness of Fit Considerations

Performing a true validation of any software reliability model that estimates N, among other parameters, requires further follow-up collection of error data during the operation and maintenance phases until the last error is found. Once obtained, the total software errors observed throughout the life cycle can be readily compared with its estimated value, N.

When this data is not available, as was the case here, validation of models must rely heavily upon the goodness of fit of the fitted points to the observed errors for each occasion.

Two goodness of fit measures, the Chi-square statistic ( $\chi^2$ ) and the product-moment correlation coefficient (r), were computed for all results presented in Tables 3.2.3-1 to 3.2.3-5. The Chi-square statistic is a standard measure commonly used for this purpose, and is computed as follows:

$$\chi^2 = \sum_{i=1}^k \frac{(n_i - n'_i)^2}{n_i}$$

where:

$n_i$  represents the errors observed on occasion i

$n'_i$  represents the errors estimated by the model for occasion i  
(i.e., the  $n'_i$  values are the fitted points to the observed errors),  
and

k is the number of occasions in the historical period.

In the context of its use and interpretation, the Chi-square statistic indicates whether the fitted points (estimated by the model) deviate from the observed data by more than random or chance error. By comparing the observed value of  $\chi^2$  with the critical value (as determined from a table of the  $\chi^2$  distribution function) for a given probability level, a determination can be made whether or not the fitted points differ only by chance from the observed errors.

The correlation coefficient ( $r$ ), on the other hand, provides information about the goodness of fit of the models from a different perspective. The value of  $r$ , which ranges from ( $0 \leq r \leq 1$ ), indicates how predictable the observed data are from the fitted points. If the predictability of historical errors is high (close to 1) rather than low (closer to 0), future error occurrences may be accurately predicted. When  $r$  is high, more confidence is placed in the validity of the parameter estimates. The values of  $r$  for all results presented in this section were computed as follows:

$$r = \left( 1 - \frac{s_n'^2}{s_n^2} \right)^{1/2} \quad (3.1)$$

where:

$s_n'^2$  is the variance of the errors of estimate; i.e., the residual error variance due to the lack of fit of the estimated points  $n_i'$  from the observed values  $n_i$ , and

$s_n^2$  is the variance of the observed errors.

Computational formulas for these variances are:

$$s_n^2 = \frac{\sum_{i=1}^k (n_i - n'_i)^2}{k-2} \quad (3.2)$$

$$s_n^2 = \frac{\sum_{i=1}^k (n_i - \bar{n})^2}{k-1} \quad (3.3)$$

where:

$\bar{n}$  is the mean of the observed error data.

The value of  $r$  in (3.1) is seen, therefore, to depend on the ratio of the two variances. For example, when the model provides a poor fit to the data, the residual error variance is large, the ratio of the variances approaches 1, and  $r$  approaches zero. When a good fit is obtained, the residual error variance is very small (hence, the ratio tends toward zero), indicating good agreement between the fitted points and the observed data (i.e.,  $r$  approaches 1). The value of  $r^2$  is also provided in all results presented here to serve as a direct measure of the percent of variation in the observed data that is "explained", or accounted for, by using the estimated values ( $n'_i$ ).

Throughout this analysis, determining whether or not the model provided a good fit to the observed data was exclusively based on the values of  $r$  and  $r^2$ . The goodness of fit of the model, or accuracy of prediction, was based on the following ranges of  $r$  and  $r^2$ :

<u>Goodness of Fit</u>	<u>Accuracy of Prediction</u>	<u><math>r</math></u>	<u><math>r^2</math></u>
Good	Average to High	0.71 to 1.00	0.50 to 1.00
Poor	Low to Moderate	0.00 to 0.70	0.00 to 0.49

The critical values of  $\chi^2$  and r at the 5 percent level of significance for each of the projects are as follows:

<u>Project</u>	<u>No. of Occasions</u>	<u>Degrees of Freedom</u>	<u>Critical Value of <math>\chi^2</math></u>	<u>Critical Value of r</u>
1	12	10	18.31	.576
2	35	33	43.77 (df=35)	.325 (df=35)
3	13	11	19.68	.553
4-7	15	13	22.36	.514

### 3.2.4 Discussion of Results

A summary of the results is presented in this section.

#### 3.2.4.1 Predictability of Software Errors from SPR Data

For Projects 1, 2, and 3 the number of software errors on each occasion were found to be highly predictable from the number of SPRs. Regression and correlation statistics were as follows, considering software errors to be a simple linear function of SPRs:

	<u>Project 1</u>	<u>Project 2</u>	<u>Project 3</u>
Intercept	7.24	0.24	-0.76
Slope	0.43	0.92	0.75
$r_2$	0.85	0.99	0.99
$r^2$	0.72	0.99	0.99

The high correlation obtained suggests that either software errors or SPRs could be estimated with reasonable accuracy using a simple ratio of SPRs to software errors, or vice versa. Since the focus of the models is based on software errors, no extensive discussion of the results for SPRs is presented here, with the exception of the following remarks.

The average number of SPRs written per software error for each project was 2.37, 1.04, and 1.33, respectively, for Projects 1, 2, and 3.

Regression statistics considering SPRs as a simple linear function of software errors were as follows:

	<u>Project 1</u>	<u>Project 2</u>	<u>Project 3</u>
Intercept	129.41	-0.20	2.81
Slope	1.65	1.08	1.32

Again, given the high correlations obtained between errors and SPRs for each project, it is not surprising that the average ratio of SPRs to errors is close to the slopes for each project. When just the ratios themselves were multiplied by the model estimate of N for software errors to give an estimate of N for SPRs, close agreement was found, in all cases, when compared with the model estimate of N for SPRs. Closer agreement was obtained in some cases using the regression equation to estimate N for SPRs using the model estimate of N for software errors as the predictor. Based on these results it was expected that the goodness of fit of the model to either software errors or SPRs should be very similar within each project. In general, the goodness of fit of the models to the software error data was either equivalent or only slightly better than that obtained using the SPR data.

There is, then, for these projects' data an indication that the assumptions used in the model for software errors apply equally well for symptoms of errors (i.e., SPRs). Therefore, one could analyze either and generalize to the other. The remainder of this validation section focuses on software errors, but would apply also to software problem reports.



#### 3.2.4.2 Goodness of Fit of Models

In general, the three model versions (binomial, Poisson, and least squares) provided equivalent fits to the observed errors in all cases.

For Projects 1, 2, and 3, given all the assumptions made about the data, each of the models provided a mixture of poor to very good fits to the observed errors, when variable testing was considered. As expected, the fits were not as good under the assumption of constant testing, except for Project 1. This result suggests that the data collected for CPU time for Project 1 do not really reflect the total effort expended in error detection. In other words, it might suggest the inappropriateness of CPU time alone as a measure of effort.

For Projects 4 to 7, the fits were moderate to good with values of  $r$  ranging between a low of  $r = .66$  for Project 7 to a high of  $r = .84$  for Project 5. Had measures of test effort been available for these projects, it is expected that improved fits of the model to the data would have been obtained.

In summary, the models provided reasonable fits to the data. For those projects having better fits than others, it is considered that the higher percentage of errors already found may be an important influence. For projects where poor fits were obtained, it was impossible to determine to what extent the data, with its attendant assumptions, was the cause.

#### 3.2.4.3 Comparison of Model Estimates of N

In general, some differences of practical significance did result from the application of one model version versus the other. However, for the samples studied thus far, no conclusions can be drawn as to the superiority of one approach over another. Of far more practical significance are differences in the estimates of  $N$  caused by the way in which data

are initially collected that later require questionable assumptions for their analysis.

In six out of eight cases for Projects 1, 2, and 3, the Poisson tended to yield the lowest estimate of N, with larger estimates respectively being obtained from the binomial and least squares model versions.

For Projects 4 through 7, the lowest estimate of N was always given by the least squares model, followed by larger estimates respectively obtained for the Poisson and binomial versions.

All estimates of N appeared reasonable given the cumulative number of historical errors observed. "Reasonable" here is meant to imply two aspects of the acceptability of the estimate. The first aspect deals with the statistical acceptability; i.e., estimates of N that are smaller than the errors already observed would be considered unacceptable or invalid statistical solutions. The second aspect concerns the operational acceptability or usefulness of these estimates from the project manager's perspective. If these estimates had been considerably larger than the errors observed (e.g., 5 to 10 times as large), particularly given the apparent extensiveness of testing that the software for these projects had undergone, the project manager would not be likely to accept the results as being very useful.

#### 3.2.5 Conclusions Based on Analysis of Historical Data

The following conclusions were drawn based on the analysis of data from Projects 1 to 7:

- (1) There were no significant differences in the goodness of fit among the binomial, Poisson, and least squares model versions.

- (2) In most cases, improved fits of the model were obtained under the assumption of variable testing.
- (3) In cases where poor fits of the model were obtained or differences of practical significance in the estimate of N were noted, the results were viewed as providing an indication of:
- the sensitivity of the model to the assumptions made about the data; and
  - the lack of thoroughness of testing of the systems being analyzed.
- (4) The results and conclusions presented are considered applicable to both software errors and software problem reports for the samples analyzed.

In general, the results point to the need for future data collection efforts to collect error history and test data that meet as closely as possible the model data requirements and assumptions (see Section 5). The model does show a great deal of promise to apply to future software development projects, especially when appropriate data are collected.

### 3.2.6 Model Validation by Means of Simulation

During the performance of this contract, considerable attention was given to achieving a more complete understanding of the model's behavior and characteristics under known conditions of software development and test. A software error data simulator program was developed and used as an analytical tool which assisted us in achieving this required understanding.

#### 3.2.6.1 Purpose of Data Simulator

The data simulator program was developed to assist in studying the effects of random error on the accuracy of the model parameter estimates. These effects are studied using software error data generated by the simulator, according to user-specified conditions of development and testing. Some background describing the role of simulation in our model validation studies is discussed in the following paragraphs.

Experiences in applying the models to ongoing projects, as well as to historical error data, have shown good indications of fit to a wide variety of software development error data. However, questionable assumptions (as made for Projects 1 to 7) have been necessary about the data prior to many of the analyses, simply because relevant information about how the software system was tested was either not collected or not available.

On occasions when a poor fit of the model to the historical data was obtained, it became difficult to determine if this lack of fit was due to the model or to the data, particularly if assumptions had to be made about the data. Ideally, when examining the behavior and characteristics

of the model, it should be possible to completely put aside questions concerned with the validity and quality of the error data. To do this, error data needs to be generated according to user-specified development and testing conditions, and in a way that closely meets the assumptions of the model.

It is expected in the real world of development and testing that error data will always have some amount of statistical error in it that keeps it from perfectly meeting all the assumptions of the model. An analytical tool was needed, then, that would allow us to generate the required, near-perfect error data according to predetermined software error, development and test conditions. The data simulator was the tool developed to meet this need.

#### 3.2.6.2 Simulation Approach

The approach to model validation using simulation was as follows. The software test conditions to be studied would first be identified. For example, consider that we wish to examine model results for a system that has been incrementally under test for five occasions and whose known values of  $N$ ,  $q$ , and  $\alpha$  are 1000, 0.2, and 1.0, respectively. Consider also that the test effort on each occasion was constant (say two units of testing per occasion). Perfect error data would then be generated according to the conditions specified, using equation (2.6) for the expected value of errors for the system. The expected value and standard deviation for each occasion would then be used to generate various samples of random error data normally distributed about the perfect values. Each of the models would then be run to estimate  $N$ ,  $q$ , and  $\alpha$  for each random sample of simulated error data. An assessment was then made of the accuracy of the model estimates of  $N$ . A summary of this

approach is presented in Figure 3.2.6.2-1 for the example conditions specified above.

### 3.2.6.3 Test Conditions Studied and Results

Admittedly there are a large number of interesting test conditions that could be examined using the approach outlined. For any condition examined, the simulation approach was found to be costly, both with respect to CPU utilization and time required to run the samples for each model version. For the purpose of this study, the results presented focus on the accuracy of estimating the parameter  $N$ .

#### Test Condition 1

An important problem initially considered was to determine the accuracy of the model estimates of  $N$  when the efficiency of testing was high and a large percentage of the total errors had already been removed. Earlier validation results had suggested that the thoroughness of testing of the system (as reflected by a high percentage of errors found relative to the estimate of  $N$ ) may influence the goodness of fit, and hence the accuracy of the model estimates. This test condition was simulated as follows:

- Test Condition 1 - Assume the system is tested (an equal amount) over ten occasions with  $N = 1000$ ,  $q = 0.2$ , and  $\alpha = 1.0$ .

Ten random samples were generated according to this condition. The errors detected after ten occasions was high in each sample, ranging from 87 percent to 94 percent of the value of  $N$  (1000). The percent average

Figure 3.2.6.2-1. SIMULATION APPROACH

---

Problem: Examine the effects of random error on model estimates

Approach:

- Identify software test conditions to be studied
- Generate perfect error data according to these conditions. For example, given a system incrementally tested a constant amount over five occasions with:

$N = 1000, q = 0.2, \alpha = 1.0$

<u>w<sub>i</sub></u>	<u>t<sub>i</sub></u>	<u>Errors</u>
0.2	2.0	72.00
0.4	2.0	118.08
0.6	2.0	219.57
0.8	2.0	212.53
1.0	2.0	136.02

- Generate normally distributed samples of random error data about perfect values; e.g.,

Simulated Random Errors

67.27  
105.70  
222.42  
215.73  
151.29

- Run models using simulated data and assess accuracy of estimates
-

error in the model estimate of N over the ten samples for each model version was as follows:

<u>Model Version</u>	<u>Percent Average Absolute Error</u>
Poisson	2.5%
Binomial	2.6%
Least Squares	2.7%

The goodness of fit for all samples using each model was very good as expected, with values of  $r = .99$  in all cases. These results essentially indicate that for the test conditions specified, with a high percentage of errors already removed, approximately 97 percent accuracy was achieved with each model version. Results similar to these were also obtained when the test conditions were changed to reflect variable testing over all occasions.

#### Test Condition 2

A second test condition was set-up to examine how many data points of error history are needed for the model to provide reasonably accurate estimates of N. This issue is closely related to the first test condition studied, in that an increase in the number of data points should correspondingly reflect an increased percentage of detected errors which should, in turn, lead to improved accuracy in the estimation of N.

For test condition 2 the following assumptions were made:  $N = 5000$ ,  $q = 0.02$ ,  $\alpha = 1.0$ , and 100 test occasions were specified, wherein the entire system (i.e.,  $w_i = 1$ ) was tested equal amounts ( $t_i = 1$ ) over all occasions. Five samples of 100 random error observations each were generated. Using only the binomial model, estimates of N and q were then obtained for each sample using 100 data points, then 99 points, 98, etc., all the way down to using just the first two or three points. In many cases when only a few points were used, the model did not converge



to a solution. The results are summarized by observation interval and are presented in Figure 3.2.6.3-1.

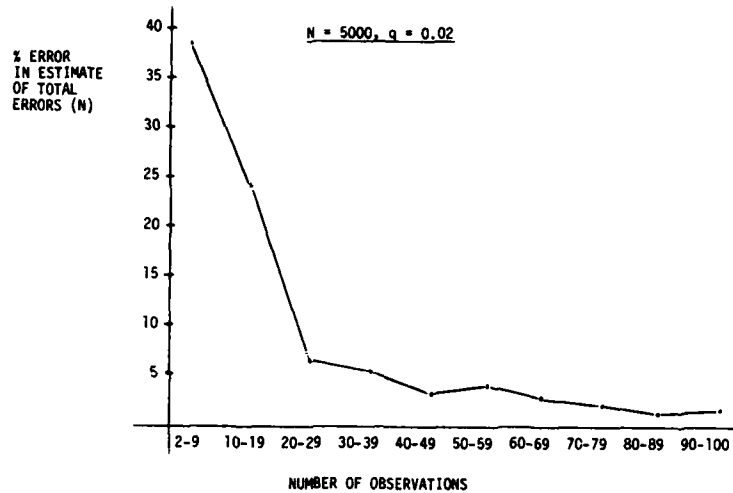
The results of this analysis, applicable to the test conditions examined thus far, indicated the following:

- Random fluctuations in the data did significantly influence the amount of error observed in the model estimate of  $N$  when ten or fewer observations were analyzed.
- An accuracy of greater than 90 percent was achieved when twenty or more observations were used.
- An ideal accuracy of 97 percent to 98 percent was obtained when approximately 75 percent of the total errors were detected.
- For the test condition examined, the goodness of fit of the model, and hence the accuracy of the estimate of  $N$ , are unsatisfactory for less than ten observations.

In an attempt to verify these results using a different but comparable set of test conditions, simulated error data was generated for 50 data points with  $N = 5000$ ,  $q = 0.0100505$ ,  $\alpha = 1.0$ , and all  $t_i$  values set equal to two. By doubling the test effort as indicated and using the value of  $q$  specified, the effective  $q_i$  being examined is 0.02, as used in the earlier test condition.

As before five random samples were examined, with the binomial model being run first with 50 points, then 49, 48, ...etc., for each sample. The results obtained using 50 observations were compared with the previous sample results obtained using 100 observations; samples with 49 data

Figure 3.2.6.3-1. SIMULATION RESULTS FOR TEST CONDITION 2



Percent Average Absolute Error as a Function of Number of Data Collection Points

Observations	Error	Ranges of $r^2$	Range for Percent of Errors Detected
2-9	37.6(a)	.07-.27	6-16
10-19	24.6(b)	.22-.69	18-32
20-29	6.5	.72-.80	33-45
30-39	5.1	.81-.85	46-55
40-49	3.6	.85-.87	56-64
50-59	4.0	.87-.90	65-71
60-69	3.4	.90-.92	72-77
70-79	2.7	.92-.93	77-82
80-89	2.1	.93-.94	82-86
90-100	2.3	.94-.95	86-89

(a) No. Samples = 20

(b) All following results based on 50 samples per observation interval.

points were compared with previous sample results using 98 data points, and so on. Essentially, no significant differences were obtained in the results as previously stated for this comparable set of test conditions.

#### 3.2.6.4 Conclusions Based on Analysis of Simulated Data

Validation of reliability models by means of simulation as described here is a feasible and meaningful alternative to validation using historical error data. This is particularly true when data have not been collected according to model data requirements and assumptions, or questionable assumptions have had to be made about existing data in order to analyze it. The results presented here show strong indications of the ability of the models examined to fit actual software error data that are collected in a manner which maximizes, to the extent possible, the close agreement of the data with the model assumptions. Lastly, we conclude that further work needs to be done in this area, wherein the effects of systematic and simultaneous variation of parameters ( $N$ ,  $q$  (or  $\phi$ ),  $\alpha$ ,  $t_i$ ,  $w_i$ ) can be examined.

### 3.3 OPTIMUM TIME INTERVAL FOR DATA AGGREGATION

In many instances in statistical analysis, it is necessary to aggregate data. For our purposes, that aggregation must be over time. Statistically speaking, however, whenever data are aggregated over time information is lost as a result of its reduced variability. An important area addressed by this contract effort was to determine how data aggregated over different time intervals affects the variability or accuracy of the model parameter estimates and goodness of fit.

A simulation approach, similar to that described in Section 3.2, was used to address this problem. The use of historical data to address this issue would be questionable, since it has been shown to violate model assumptions.

Using the simulation approach, a test condition was initially specified of a system constantly under test. An equal amount ( $t_i=1$ ) of testing effort was specified over 100 occasions with  $N = 5000$ ,  $q = 0.02$ , and  $\alpha = 1.0$ . Aggregations of software error data were then obtained by varying the number of points within each time interval. Using the values of  $N$ ,  $q$ , and  $\alpha$  as specified, random samples of aggregated error data were generated for 2, 4, 5, 10, and 20 point data aggregations.

What was being accomplished here was the successive aggregation of data points taken from an initial sample containing 100 observations of errors, generated with  $N = 5000$ ,  $q = 0.02$ ,  $\alpha = 1.0$ ,  $t_i = 1.0$ , and  $w_i = 1.0$ . For example, the two point aggregated sample contained 50 observations similarly generated with the exception that all the  $t_i$  values were  $t_i = 2$ . The four point sample contained 25 observations, generated with  $t_i = 4$ ; and so forth for the remaining aggregations. Ten random samples were generated for each set of data aggregations (1, 2, 4, 5, 10, 20). The binomial model was then run against all 60 samples to determine the estimated values of  $N$  and  $q$  for the test conditions specified.

The results of this analysis are presented in the following table, which describes the percent average absolute error in the estimated value of N for each set of 10 aggregated data samples.

<u>Data Points Aggregated</u>	<u>No. of Observations per Sample</u>	<u>Percent Average Absolute Error</u>
1	100	1.8%
2	50	1.5%
4	25	0.9%
5	20	1.2%
10	10	1.6%
20	5	1.6%

The goodness of fit of the model was very good in all cases, with values of  $r^2$  ranging from .93 to .99 among the 60 samples analyzed. The percentage of errors found in all samples was essentially the same, ranging from 85 percent to 89 percent of  $N = 5000$ . These results, limited to the test conditions examined, indicate that when random fluctuations are present, the way in which the data are aggregated has no impact on the model estimates. Thus, when the assumptions of the model are met, the level of aggregation has no influence.

These results apply only to the test conditions specified, which include the values of the parameters as well as the percentage of the total errors which are assumed or detected. A more thorough study would be required before more general conclusions could be drawn. Further study would be directed at a systematic variation of the model parameters, as well as the percentage of errors detected.

### 3.4 TIME TO A SPECIFIED NUMBER OF REMAINING ERRORS

There are two aspects to the problem of estimating time to a specified number of errors. One of these is the detection time and the other is repair time. We first discuss the problem of detection time and then discuss the problem of repair time.

In a system where no additional errors are made in attempting to correct errors, the problem is to estimate the total number of errors remaining at time  $T_i$ , denoted as  $N_{ri}$ , subtract the specified number  $N_{rs}$ , and then estimate the time to detect the difference  $N_{ri} - N_{rs}$ . However, in order to have a specified number remaining, we must detect (and correct) an additional number due to reinserting errors in the attempt to correct other, known errors. Let us denote the probability of detecting an individual error as  $q$ , and the probability of correcting the error once it is found as  $\alpha$ .

The process we are modeling may be depicted in a tree structure, as shown in Figure 3.4-1. The nodes of the tree represent the number of errors remaining and the links represent probabilities of transition. At each stage we are depicting the number of errors remaining, given that one has been detected. This figure terminates at Stage 3, but subsequent stages would be similar, except that we terminate (reach an absorbing state) a branch of the tree when the number remaining is  $N_{rs}$ . The problem then becomes the estimation of transition times and probabilities to reach the specified number,  $N_{rs}$ .

#### 3.4.1 Formulation

Let us denote the time associated with the detection of error  $n_i$  as  $t_i$ . According to our assumptions that errors found are proportional to errors remaining, and the error probability is constant and a function of time, the expected number of errors on the  $i$ th plus first  $(i+1)$  test occasion is:

$$n'_{i+1} = N_{ri} \left[ 1 - (1-q)^{t_{i+1}} \right] \quad (3.4)$$

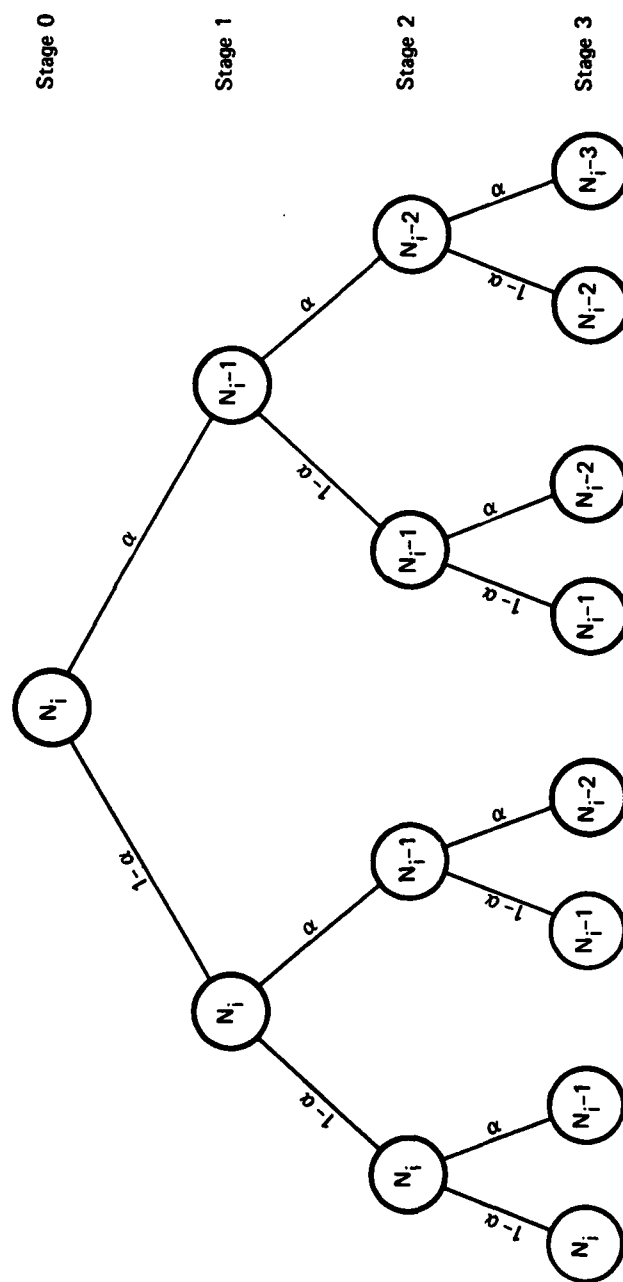


Figure 3.4-1. REPRESENTATION OF THE ERROR DETECTION AND CORRECTION PROCESS

Without loss of generality, we may consider the errors one at a time, assume that one is found and solve for  $t_{(N_{ri}-1)}$ , the time required to reduce the remaining errors by one. We have, then, by setting the expected number equal to one:

$$(1-q)^{t_{i+1}} = 1 - \frac{1}{N_{ri}}, \text{ or}$$

$$t_{(N_{ri}-1)} = t_{i+1} = \frac{1}{\ln(1-q)} \ln \left( 1 - \frac{1}{N_{ri}} \right) \quad (3.5)$$

The quantity  $T_{i \rightarrow s}$ , defined as the total detection time required to reduce the errors remaining from  $N_{ri}$  to  $N_{rs}$  under perfect debugging (i.e.,  $\alpha=1$ ), is then

$$T_{i \rightarrow s} = \sum_{j=1}^{(N_{ri}-N_{rs})} t_{(N_{ri}-j)} = \frac{1}{\ln(1-q)} \ln \frac{N_{rs}}{N_{ri}} \quad (3.6)$$

This quantity will be used later. We now turn to the problem of the number of detections required and the total time required to reduce the errors to the specified number under imperfect debugging conditions. Define the random variable  $x$  as the total number of detections required. Then  $x$  will range from  $(N_{ri} - N_{rs})$  to infinity, and its probability distribution will be determined by  $\alpha$ , the probability of correcting an error.

The following table shows the distribution of  $x$  and the times associated with various values of  $x$ . In order to facilitate discussion let us define the minimum value of  $x$  as  $r = N_{ri} - N_{rs}$ .



<u>x</u>	<u>Probability of (x)</u>	<u>Time to Detect x</u>
r	$\alpha^r$	$\alpha^r T_{i \rightarrow s}$
r+1	$\binom{r}{r-1} \alpha^r (1-\alpha)$	$\binom{r+1}{r} \alpha^r (1-\alpha) T_{i \rightarrow s}$
r+2	$\binom{r+1}{r-1} \alpha^r (1-\alpha)^2$	$\binom{r+2}{r} \alpha^r (1-\alpha)^2 T_{i \rightarrow s}$
r+3	$\binom{r+2}{r-1} \alpha^r (1-\alpha)^3$	$\binom{r+3}{r} \alpha^r (1-\alpha)^3 T_{i \rightarrow s}$
$\vdots$	$\vdots$	$\vdots$
r+j	$\binom{r+j-1}{r-1} \alpha^r (1-\alpha)^j$	$\binom{r+j}{r} \alpha^r (1-\alpha)^j T_{i \rightarrow s}$

These probabilities and times can be calculated from the negative binomial distribution.

The general form of the negative binomial probability distribution is:

$$f(x; k, \theta) = \binom{x-1}{k-1} \theta^k (1-\theta)^{x-k}$$

The random variable x represents the number of trials required to achieve k successes, where an individual success has a probability  $\theta$  of occurring. The mean and variance of this distribution are:

$$E(x) = \frac{\theta}{k}$$

$$V(x) = \frac{\theta}{k} \left( \frac{1}{\theta} - 1 \right)$$

In our example, both x and the time required to detect x can be represented by the negative binomial. In the case of x, the number of trials (detections) required to achieve r successes (error reductions), it is

distributed as a negative binomial with  $k=r$ . The time required to achieve  $x$  successes is obtained by summing the values in the final column. Thus:

$$\begin{aligned}
 T'_{i \rightarrow s} &= T_{i \rightarrow s} \sum_{j=0}^{\infty} \binom{r+j}{r} \alpha^r (1-\alpha)^j \\
 &= \frac{T_{i \rightarrow s}}{\alpha} \sum_{j=0}^{\infty} \binom{r+j}{r} \alpha^{r+1} (1-\alpha)^j \\
 &= \frac{T_{i \rightarrow s}}{\alpha} \tag{3.7}
 \end{aligned}$$

since the summation represents a probability distribution which sums to 1, i.e., a negative binomial with parameters  $r$  and  $\alpha$ .

### 3.4.2 Example

An example of the application of these formulas can be seen by the following:

Number of errors remaining:	$N_{ri} = 10$
Specified number of errors remaining:	$N_{rs} = 5$
Probability of error detection:	$q = .02$
Probability of error correction:	$\alpha = .90$

Then:

$$\begin{aligned}
 t_9 &= \frac{1}{\ln(.98)} \ln \left( 1 - \frac{1}{10} \right) \\
 t_8 &= \frac{1}{\ln(.98)} \ln \left( 1 - \frac{1}{9} \right) \\
 &\vdots \\
 t_5 &= \frac{1}{\ln(.98)} \ln \left( 1 - \frac{1}{6} \right)
 \end{aligned}$$

$$T_{i \rightarrow s} = \sum_{j=1}^5 t_{10-j} = \frac{1}{\ln(.98)} \ln \frac{5}{10} = 34.31$$

and

$$T'_{i \rightarrow s} = \frac{34.31}{.9} = 38.12$$

This approach is actually independent of the assumptions made in deriving the quantity for time in (3.6). The assumptions could be replaced by any corresponding set and inserted into (3.7). For example, one common assumption utilized in software reliability theory is that the number of errors found is proportional to the number remaining in such a way that:

$$t(N_{ri}-1) = \frac{1}{N_{ri}\phi}$$

In such a case, for the present example, we would have ( $\phi=.02$ ):

$$t_9 = \frac{1}{10(.02)} = 5$$

$$t_8 = \frac{1}{9(.02)} = 5.56$$

$$t_7 = \frac{1}{8(.02)} = 6.25$$

$$t_6 = \frac{1}{7(.02)} = 7.14$$

$$t_5 = \frac{1}{6(.02)} = 8.33.$$

Then

$$T_{i \rightarrow s} = 32.28$$

and from (3.7)

$$T'_{i \rightarrow s} = 35.87$$

Equation (3.7) may also be used to incorporate correction time. The times for detection and for correction may be derived by any theory, or combination of theories. For example, we might assume that detection

time is determined as our model dictates, and that correction time is a constant or independent of the number of errors remaining. Or, on the other hand, we might assume that correction time is proportional to the number of remaining errors.

### 3.4.3 Combining Detection and Correction Times

To illustrate the application of the approach to correction time as well as detection time, suppose we had the following situation:

Number of remaining errors	$N_r = 5$
Specified number of remaining errors	$N_s = 3$
Probability of correcting a detected error	$\alpha = .9$
Error detection rate	$q_d = .02$
Error correction rate	$q_c = .05$

We further assume that the times to detection are given by (3.6) and that the times to correction increase in inverse proportion to the number of errors remaining. Then, for detection we have:

$$T_{5 \rightarrow 3, d} = \frac{1}{\ln(.98)} \ln \frac{3}{5} = 25.3$$

The correction times are:

$$T_{5 \rightarrow 3, c} = \frac{1}{5(.05)} + \frac{1}{4(.05)} = 9.0$$

Combining the two we have:

$$T_{t \rightarrow 3} = 25.3 + 9.0 = 34.3$$

Using (3.7) we have

$$T'_{5 \rightarrow 3} = \frac{34.3}{.9} = 38.1$$

If we had assumed that the detection times as well as correction times were inversely proportional to the number of errors remaining we would have had:

$$T_{5 \rightarrow 3, d} = 22.5$$

$$T_{5 \rightarrow 3, c} = 9.0$$

$$T_{5 \rightarrow 3} = 31.5$$

$$T'_{5 \rightarrow 3} = 35.0$$

Table 3.4-1 shows the times to reduce errors to a pre-specified number as a function of the assumptions made as well as the parameters. Under time to detect, assumption (1) is that detection time increases according to (3.5), while assumption (2) is that detection time is inversely proportional to the number of errors remaining. Time to correct is also assumed to be inversely proportional to the errors remaining.

Note in Table 3.4-1 that no calculations are made for the situation of reducing the number of errors to zero. In this case Assumption (1) leads to an estimate of infinity. That is, to find the last error in a program which is known to contain some errors would require an infinite amount of time. Thus, one can never be sure that the last error has been found.

In general, to reduce the number of remaining errors  $N_{ri}$  to a specified number,  $N_{rs}$ , under assumption (2) is:

$$T_{i \rightarrow s} = \sum_{j=1}^{(N_{ri} - N_{rs})} t_{(N_{ri} - j)} = \frac{1}{q} \sum_{j=1}^{(N_{ri} - N_{rs})} \frac{1}{(N_{ri} - j)} \text{ and}$$

Table 3.4-1. TIMES TO REDUCE REMAINING ERRORS TO A SPECIFIED NUMBER

$N_r=10$ ,  $\alpha=.9$ ,  $q_d=.02$ ,  $q_c=.05$

Remaining Errors Specified ( $N_s$ )	Assumption	Assumption	Time to Correct ( $T_{10 \rightarrow s, c}$ )	Total Time Assumption	
	(1)	(2)		(1)	(2)
9	5.79	5.56	2.22	8.01	7.78
8	12.27	11.73	4.69	16.96	16.42
7	19.62	18.67	7.47	27.09	26.14
6	28.09	26.61	10.64	38.73	37.25
5	38.12	35.87	14.35	52.47	50.22
4	50.39	46.98	18.79	69.18	65.77
3	66.22	60.87	24.35	90.57	85.22
2	88.52	79.39	31.75	167.91	111.14
1	126.64	107.16	42.87	169.51	150.03

$$T'_{i \rightarrow s} = \frac{1}{\alpha q} \sum_{j=1}^{(N_{ri} - N_{rs})} \frac{1}{(N_{ri} - j)}$$

For large values of  $N_r$  the calculation using this formula is not formidable with a digital computer. However, for the sake of overall economy, an approximation may be used.

The approximation is:

$$\sum_{i=1}^n \frac{1}{i} = \ln n + c + \frac{1}{2n} - \frac{1}{12n^2}$$

where  $c = .5772$  (Euler's constant)

Putting this equation into our notation we have

$$T'_{i \rightarrow s} = \frac{1}{\alpha q} \sum_{i=1}^{N_{ri}} \frac{1}{i} - \sum_{i=1}^{N_{rs}} \frac{1}{i}$$

$$T'_{i \rightarrow s} = \frac{1}{\alpha q} \left[ \left( \ln \frac{N_{ri}}{N_{rs}} \right) + \left( \frac{1}{2N_{ri}} - \frac{1}{12N_{ri}^2} \right) - \left( \frac{1}{2N_{rs}} - \frac{1}{12N_{rs}^2} \right) \right] \quad (3.8)$$

The accuracy of this approximation may be evaluated by using it for the detection times in Table 3.4-1. For comparison, the actual values are reproduced from that table.

$$N_r = 10, \alpha = .9, q_d = .02$$

<u>Errors<sub>N<sub>s</sub></sub> Specified</u>	<u>Actual Time</u>	<u>Time Estimated Using Equation (3.8)</u>
9	5.56	5.56
8	11.73	11.73
7	18.67	18.67
6	26.61	26.61
5	35.87	35.87
4	46.98	46.98
3	60.87	60.87
2	79.39	79.41
1	107.16	107.50

Thus, for even small numbers of errors the approximation is in excellent agreement with the actual.

For larger values of  $N_r$  the approximation will be even more accurate. The following table shows the time to reduce the original errors of 100 to various values, with  $\alpha=.9$  and  $q_d=.02$ .

<u>N<sub>s</sub></u>	<u>Detection Time</u>
90	5.82
80	12.33
70	19.70
60	28.19
50	38.23
40	50.49
30	66.24
20	88.31
10	125.47
1	232.97



### 3.5 MODEL FORMULATION WITH VARIABLE DETECTION PROBABILITY

In this model we allow for a probability of detection which first increases by a constant amount until a peak is reached, at which time the probability becomes constant. This formulation is roughly equivalent to a learning curve. Translated into the software testing world, it would be equivalent to a testing team beginning to test on a system with which they are unfamiliar, but with increasing experience becoming increasingly efficient until a plateau in efficiency is reached.

According to this formulation  $q$  increases on each test occasion by a constant amount  $c$ , until it reaches a maximum of  $q + (j-1)c$  on occasion  $j$ . This maximum value then applies to the remaining  $(j+1)$  to  $k$  additional occasions. For this derivation we have assumed that time intervals are equal, the total system is constantly under test, and the probability of correcting a detected error is 1. The reason for making these assumptions for the derivation is that there is a danger of over specification of the model. That is, the increased probability might very well be due to some of these factors. If they were actually incorporated into the model, we would then be accounting for them twice, thus overspecifying them.

The acid test of the generalization to include a variable detection probability is how well does it fit actual data. Thus, a first step in the use of the equations resulting from the model would be an application to actual data.

Three equations result for this formulation. They would be used to estimate the three parameters:  $N$ , total number of errors in the software;  $q$ , the probability of detection; and  $c$ , the constant increase in  $q$ .

The equations are:

$$\frac{N - N_{q+k}}{N} = \left[ \prod_{i=0}^{j-1} (1 - q - ic) \right] \left[ 1 - q - (j-1)c \right]^k \dots \quad (3.9)$$

$$\sum_{i=1}^j \frac{n_i}{q + (i-1)c} + \frac{1}{q + (j-1)c} N_k = \sum_{i=1}^j \frac{N - N_i}{1 - q - (i-1)c} + \frac{1}{1 - q - (j-1)c} \sum_{i=1}^k (N - N_{j+i}) \dots \quad (3.10)$$

$$\sum_{i=1}^j \frac{i n_i}{q + (i-1)c} + \frac{j}{q + (j-1)c} N_k = \sum_{i=1}^j \frac{i (N - N_i)}{1 - q - (i-1)c} + \frac{j}{1 - q - (j-1)c} \sum_{i=1}^k (N - N_{j+i}) \dots \quad (3.11)$$

where:

$N$  = Estimate of total error population

$N_i$  = Total observed errors through occasion  $i$

$N_k$  = Total observed errors on occasions  $j + 1$  through  $j + k$

$n_i$  = Observed errors on occasion  $i$

$q$  = Estimate of error detection probability

$c$  = Increment in error detection probability estimate

$j$  = Occasion on which probability of detection reaches a  
maximum

$k$  = Total number of occasions after  $j$ .

The occasion ( $j$ ) on which the probability of detection reaches a maximum should be determined from a visual inspection of the graph of the number of errors versus time.

These equations have not been applied to actual data, so there is no experience to draw on. It should be noted that the detection probability does not necessarily reach a maximum on the same occasion as the maximum number of detected errors. As a matter of fact, the detected errors may be at a maximum on the first occasion with the probability reaching a peak at some later time. To guard against this eventuality, several values of  $j$  should be tried and that value chosen that gives the best fit of observed to actual data.

### 3.6 NARROWING THE RANGE ESTIMATES OF MODEL PARAMETERS

When the assumption is made that the probability of perfect error corrections is less than one (i.e.,  $\alpha < 1$ ) and the test effort is constant over all occasions (e.g., all  $t_i = 1$ ), the three model equations (for any model version) in the three unknowns  $N$ ,  $q$  (or  $\phi$ ), and  $\alpha$  are not independent.

The resultant equations determine limits of the three parameters and the reliability estimate derived from them. These limits are dependent such that the specification of any two parameters uniquely determines the third. For example, for a given set of observed data, the maximum value of  $N$  implies that  $q$  or  $\phi$  is a minimum and  $\alpha$  is at its maximum value. Whereas the minimum value of  $N$  implies that  $q$  or  $\phi$  is a maximum and that  $\alpha$  is a minimum. It should be remembered that  $N$  represents the original number of errors in the software.

The dependence of the three equations arises from the assumption that  $\alpha$  is a constant and is proportional to the number of errors detected on a particular occasion. There are several ways of analyzing this dependence. One is to provide an estimate of  $\alpha$  from previous experience. Another is to change the assumptions so that independent equations may be derived. Still another is to consider the range estimates as a valid output of the model, but to examine the equations more closely so that the range estimates can be narrowed. The approach taken to this problem was to narrow, if feasible, the range estimates of the model parameters.

The analysis was started considering that for any given set of error data, the natural lower limit for N is as follows:

when  $\alpha = 1$

$$N_{\min} = \sum_{i=1}^k n_i$$

when  $\alpha < 1$ ,

$$N_{\min} = \sum_{i=1}^k n_i - (1-\alpha) \sum_{i=1}^k n_i = \alpha \sum_{i=1}^k n_i$$

where:

$$\sum_{i=1}^k n_i \text{ is the total number of errors observed to date over } k \text{ occasions.}$$

Considerable attention was devoted to the study of this problem. Four alternative methods of solution to the model equations were developed and applied in an attempt to find unique solutions for N, q (or  $\phi$ ), and  $\alpha$  that were within the range estimates for these parameters. Internal software error data were used in this analysis.

The net result of this effort was that no unique solutions to the equations were found to exist under these conditions, and that the range estimates should be considered as valid output from the models.

Some practical suggestions can be made, nevertheless, with respect to specifying the ranges within which given iteration and convergence procedures (e.g., Newton-Raphson) are allowed to search for a solution to the three equations. A reasonable upper bound for N would be  $N_{\max} \leq 100N_{\min}$ . If no more than 1 percent of the errors have been removed, then testing and debugging should be the primary concerns rather than modeling.

Ranges for  $q$  (or  $\phi$ ) and  $\alpha$  are suggested as follows:

$$0 < q \text{ (or } \phi) \leq .5$$

$$.5 < \alpha \leq 1$$

The rationale for  $q$  (or  $\phi$ )<sub>min</sub> is that if errors have been detected, then clearly ( $q$ <sub>min</sub>  $\neq 0$ ). Operationally, a  $q$  (or  $\phi$ )<sub>max</sub>  $\leq .5$  is a reasonable upper limit since no one testing occasion is likely to be more efficient than to detect over 50 percent of the errors.

The bounds for  $\alpha$  are determined from a certain degree of faith in the programmers. A value of  $\alpha$  less than .5 would imply that the programmers who are debugging the detecting errors are making more errors than they are correcting. This seems unreasonable. Therefore, the lower limit is taken as .5.

### 3.7 VARIANCE AND CONFIDENCE LIMITS FOR MODEL PARAMETERS

To determine the variance and covariance estimates of the parameters  $N$ ,  $q$ , and  $\alpha$ , we define:

$$A \equiv \begin{pmatrix} \frac{\partial^2 \ell_{NL}}{\partial N^2} & \frac{\partial^2 \ell_{NL}}{\partial q \partial N} & \frac{\partial^2 \ell_{NL}}{\partial \alpha \partial N} \\ \frac{\partial^2 \ell_{NL}}{\partial N \partial q} & \frac{\partial^2 \ell_{NL}}{\partial q^2} & \frac{\partial^2 \ell_{NL}}{\partial \alpha \partial q} \\ \frac{\partial^2 \ell_{NL}}{\partial N \partial \alpha} & \frac{\partial^2 \ell_{NL}}{\partial q \partial \alpha} & \frac{\partial^2 \ell_{NL}}{\partial \alpha^2} \end{pmatrix}$$

where:

$L$  is the likelihood function for either the binomial or Poisson maximum likelihood formulations as presented in Section 2.0.

The elements of the inverse of this matrix then give the variance and covariance estimates. Thus:

$$A^{-1} \equiv \begin{pmatrix} \sigma_N^2 & \sigma_N \sigma_q \rho_{Nq} & \sigma_N \sigma_\alpha \rho_{N\alpha} \\ \sigma_q \sigma_N \rho_{qN} & \sigma_q^2 & \sigma_q \sigma_\alpha \rho_{q\alpha} \\ \sigma_\alpha \sigma_N \rho_{\alpha N} & \sigma_\alpha \sigma_q \rho_{\alpha q} & \sigma_\alpha^2 \end{pmatrix}$$

Confidence limits for the estimates of the parameters are obtained by assuming a normal distribution of the errors of estimate and using the square-roots of the diagonal elements of  $A^{-1}$  as the standard deviations. Thus, the 95 percent confidence limits for a parameter would be the estimate of that parameter  $\pm$  two standard deviations.

The equations to estimate the second derivatives of the likelihood functions were actually derived, and the elements of  $A^{-1}$  were evaluated, using project error data and parameter estimates obtained from this data. This was accomplished for the least squares as well as the binomial and Poisson maximum likelihood model versions. The resulting equations for the variance estimates of the model parameters were, in each case, quite complex. When confidence intervals for  $N$ ,  $q$ , and  $\alpha$  were evaluated using these variance estimates for a given set of data, they were found to be quite large thus having little practical value in actual model applications. Based on these results, no further work was performed in this area.



### 3.8 COMPARISON OF DISCRETE MODELS WITH WEIBULL DISTRIBUTION

The purpose of this task was to determine the form of the Weibull distribution which fits error history data, and to show the relationship of the binomial and Poisson model parameters to those of the Weibull.

The general form of the Weibull density function is as follows:

$$f(t; \alpha, \beta, \gamma) = \alpha \beta (t - \gamma)^{\beta-1} e^{-\alpha(t-\gamma)^\beta} \quad (3.12)$$

with the cumulative probability,

$$P(T < t) = F(t; \alpha, \beta, \gamma) = 1 - e^{-\alpha(t-\gamma)^\beta} \quad (3.13)$$

where:

$\alpha$  is the scale parameter  $(\alpha > 0)$

$\beta$  is the shape parameter, and  $(\beta > 0)$

$\gamma$  is the location parameter.  $(t > \gamma)$

For the Weibull model, the likelihood function is:

$$L = \prod_{i=1}^k \alpha \beta (T_i - \gamma)^{\beta-1} e^{-\alpha(T_i - \gamma)^\beta} \quad (3.14)$$

where:

$k$  is the number of observations or occasions;

$T_i$  is the cumulative time through occasion  $i$ .

Taking the logarithm of (3.14) yields:

$$\ln L = k \ln \alpha + k \ln \beta + (\beta - 1) \sum_{i=1}^k \ln(T_i - \gamma) - \alpha \sum_{i=1}^k (T_i - \gamma)^\beta \quad (3.15)$$

The maximum likelihood estimates of the Weibull model parameters are the solutions to the following set of equations, the partial derivatives of (3.15) with respect to  $\alpha$ ,  $\beta$ , and  $\gamma$ :

$$\frac{\partial \ln L}{\partial \alpha} = 0 = \frac{k}{\alpha} - \sum_{i=1}^k (T_i - \gamma)^\beta$$

$$\frac{\partial \ln L}{\partial \beta} = 0 = \frac{k}{\beta} + \sum_{i=1}^k \ln(T_i - \gamma) - \alpha \sum_{i=1}^k (T_i - \gamma)^\beta \ln(T_i - \gamma)$$

$$\frac{\partial \ln L}{\partial \gamma} = 0 = \alpha \beta \sum_{i=1}^k (T_i - \gamma)^{\beta-1} - (\beta - 1) \sum_{i=1}^k \left( \frac{1}{T_i - \gamma} \right)$$

Once estimates for the Weibull parameters are obtained, an equation is needed for the estimate of  $N$ , the total number of errors in the system. Toward this end, (3.13) is used to determine the expected number of errors ( $N'_k$ ) detected through time  $T_k$ :

$$N'_k = N \left[ 1 - e^{-\alpha(T_k - \gamma)^\beta} \right] \quad (3.16)$$

where:

$T_k$  is the cumulative time through occasion  $k$ .

Using (3.16) the estimate for  $N$  is:

$$N \approx \frac{N_k}{1 - e^{-\alpha(T_k - \gamma)^\beta}} \quad (3.17)$$

where:

$N_k$  is the cumulative number of errors observed through  $k$ .

AD-A086 334

IBM FEDERAL SYSTEMS DIV GAITHERSBURG MD  
ANALYSIS OF DISCRETE SOFTWARE RELIABILITY MODELS.(U)  
APR 80 W D BROOKS, R W MOTLEY

F/G 9/2

F30602-78-C-0346

UNCLASSIFIED

RADC-TR-80-84

NL

2 of 2

AD  
NO. 334

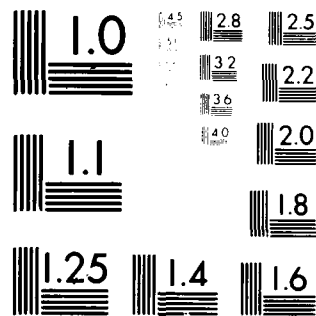
END

DATE

FILMED

8-80

DTIC



MICROCOPY RESOLUTION TEST CHART  
NATIONAL BUREAU OF STANDARDS-1963-A

This equation for N may be directly compared with that of the binomial model assuming equal time intervals. That equation is:

$$N = \frac{N_k}{1-(1-q)^k}$$

Thus, for equal time intervals, and noting that:

$$\lim_{x \rightarrow 0} (1 - x)^k = e^{-kx}, \text{ for } (0 \leq x < 1)$$

the parameters of the Weibull are related to the binomial and will give the identical estimates of N if:

$$k = \beta, \text{ and} \\ q = \alpha (T_k - \gamma).$$

The Weibull model is useful as a curve fitting tool. But theoretically and operationally, the parameters of this model do not lend themselves to direct comparison and interpretation with those of the binomial and Poisson when applied to the real world of software development and testing. Application of the Weibull model to software error data essentially indicated that all three model parameters ( $\alpha, \beta, \gamma$ ) need to be estimated before reasonable agreement with observed data is obtained.

## SECTION 4

### MODEL IMPLEMENTATION

#### 4.1 INTRODUCTION

The software reliability models described in Section 2 have been developed to support project management requirements for timely software reliability analysis during the development process. The models have been implemented in APL at IBM/FSD Gaithersburg. The purpose of this section is to briefly describe and highlight the capabilities provided by these model programs. It is also intended to further aid the reader's understanding of:

- The rationale for and usefulness of the module and system level versions of the reliability model; and
- The manner in which software error data is specified as input to these models.

#### 4.2 CAPABILITIES

When supplied with the required software error data, the reliability model programs can provide the following information about the software system under development:

Estimates of:

- The total number (N) of software errors originally present in the software system prior to testing;

- The probability ( $q$  or  $\phi$ ) of detecting any one error during a user specified unit of test effort; and
- The probability ( $\alpha$ ) of correcting an error without reinserting additional errors and exposing others to discovery.

Based on each of the estimates as previously indicated, the following additional information is provided as part of the program output:

- The model's fitted points to the observed error history data.
- A measure of the goodness of fit and accuracy of prediction between the model's fitted points and the user's software error data; a chi-square statistic and correlation coefficient are respectively provided for this purpose.
- A direct measure of the percent of variation in the observed data that can be accounted for using the model's fitted points; the squared correlation coefficient is used for this purpose.

Once estimates of  $N$ ,  $q$  (or  $\phi$ ), and  $\alpha$  are obtained from any given model run, other meaningful statistics related to software reliability can be computed by the user according to the appropriate mathematical formula or equations described in other sections of this report. Some of the statistics that can be computed using these model estimates are:

- The number of errors remaining in the software.
- Predictions of future error occurrences based on additional testing effort specified by the user.

- Measures of software reliability with and without additional testing effort.
- Probability of passing a user-specified requirement test (e.g., the probability that no more than 10 system reload type software errors will occur during a 200 hour customer acceptance test).
- Future test time required to achieve a specified reliability.

#### 4.2.1 Model Versions

Essentially, there are three different versions of the software reliability model. These versions are referred to as the binomial maximum likelihood (BML), the Poisson maximum likelihood (PML), and the least squares version (BLS). Each of these model versions has been implemented to estimate either 2 (N and q) or 3 (N, q, and  $\alpha$ ) model parameters according to the user's needs, and to process error data collected at either the system or module level. The APL model versions are referenced and identified as follows:

	<u>Module Level</u>	<u>System Level</u>
2 Parameter	BMLM2	BMLS2
	PMLM2	PMLS2
	BLSM2	BLSS2
3 Parameter	BMLM3	BMLS3
	PMLM3	PMLS3
	BLSM3	BLSS3

With the two parameter model versions, user's may specify the value of  $\alpha$  to consider when estimating N and q. When using the Poisson models  $\phi$  is estimated in place of q.



#### 4.2.2 Module Level Models

The APL module level model versions allow the user to:

- Estimate  $N$ ,  $q$ , and  $\alpha$  for each individual module or subsystem that comprise the total system. When used for this purpose, a model run will be required for each of the modules that the estimates are needed for. Once obtained, these estimates apply only to the module or subsystem for which the error data were collected and analyzed; or
- Estimate  $N$ ,  $q$ , and  $\alpha$  for the entire software system, using all available error and test data that has been collected at the module or subsystem level for this purpose. The fitted points for each module are also provided in this case.

The form of the input data required to use the models at this level is as follows:

##### MODULE LEVEL DATA

$n_{11}$	$n_{12}$	$\dots$	$n_{1j}$
$n_{21}$	$n_{22}$	$\dots$	$n_{2j}$
$\vdots$	$\vdots$		$\vdots$
$n_{i1}$	$n_{i2}$	$\dots$	$n_{ij}$
$t_{11}$	$t_{12}$	$\dots$	$t_{1j}$
$t_{21}$	$t_{22}$	$\dots$	$t_{2j}$
$\vdots$	$\vdots$		$\vdots$
$t_{i1}$	$t_{i2}$	$\dots$	$t_{ij}$
$w_1$	$w_2$	$\dots$	$w_j$

where  $n_{ij}$  represents the software errors observed over ( $i = 1, 2, \dots k$ ) test occasions for ( $j = 1, 2, \dots m$ ) modules or subsystems;  $t_{ij}$  is the testing effort expended over the  $k$  test occasions for the  $m$  modules; and  $w_j$  ( $j = 1, 2, \dots m$ ) represents the percentage of the total source lines of code that each module represents. For example, if the size of module 1 is 5,000 source lines of code (SLOC) and the total system size is 50,000 SLOC, then the weight ( $w_1$ ) for module 1 would be  $5KSLOC/50KSLOC = 0.10$ . The weights for the remaining modules are computed accordingly. The weights ( $w_j$ ) as used at the module level are considered to be constant over all test occasions.

Two notes are important to mention here. First, it is frequent that software error and test data have not been collected or are not available at lower levels other than at the total system level. The system level data in this case would simply be a one column matrix (with dimensions  $2k + 1, 1$ ) containing the  $n_i$  and  $t_i$  values for  $k$  test occasions for the system, followed by the constant value for the system weight ( $w_j$ ) to be assigned over all occasions. The module level versions can process this data under the assumption that equal portions (i.e., percent of total size) of the system have been under test on each occasion. In most cases this would be 100% of the system under test over all occasions (i.e., a weight of 1.0). If this assumption is not reasonable or realistic, then the system level data with the appropriate weights for each test occasion can be processed using the system level model versions.

Second, when the module level versions are being used to estimate  $N$ ,  $q$ , and  $\alpha$  for the entire software system using the lower level module or subsystem data, it is common that some modules will be in and out of test, in effect not being tested on all occasions. When this occurs, the values for errors ( $n$ ) and test time ( $t$ ) are set to zero for those modules on those occasions.

#### 4.2.3 System Level Models

The APL system level model versions allow the user to estimate  $N$ ,  $q$ , and  $\alpha$  for the entire software system. These versions have been developed to accomodate the situation wherein testing on each occasion is performed on and measurable at the system level, or at least at some level higher than the module level. This implies that for each test occasion the module/subsystem level error ( $n_{ij}$ ) and weight ( $w_j$ ) data have been recorded, and only one value of test effort ( $t_i$ ) for the occasion is measured and recorded.

The form of the input data required to use the models at this level is as follows:

$$\begin{array}{c} \text{SYSTEM LEVEL DATA} \\ \left( \begin{array}{cccccccc} n_{11} & n_{12} & \cdots & n_{1j} & w_{11} & w_{12} & \cdots & w_{1j} & t_1 \\ n_{21} & n_{22} & \cdots & n_{2j} & w_{21} & w_{22} & \cdots & w_{2j} & t_2 \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots \\ \vdots & \vdots & & \vdots & \vdots & \vdots & & \vdots & \vdots \\ n_{i1} & n_{i2} & & n_{ij} & w_{i1} & w_{i2} & \cdots & w_{ij} & t_i \end{array} \right) \end{array}$$

where:

$n_{ij}$  has the meaning defined for the module level;

$w_{ij} = \begin{cases} w_j & \text{if module } j \text{ was tested on occasion } i \\ 0 & \text{otherwise} \end{cases}$ ; and

$t_i =$  the test effort expended on the system during the  $i$ th occasion.

#### 4.3 MODEL PROGRAM DOCUMENTATION

A User's Guide for the software reliability models has been delivered to RADC as part of this technical contract effort. It's contained in a deliverable document dated October 1979 entitled Software Reliability Model Computer Programs and Documentation. It further contains example inputs and outputs and all the APL program functions needed to implement each of the models discussed in this report.

## SECTION 5

### DATA REQUIREMENTS FOR SOFTWARE RELIABILITY ANALYSIS

#### 5.1 REQUIREMENTS

The models described in this report have been developed and applied to meet a variety of project management reliability information needs. From project to project these needs may differ. On one project, error data may need to be collected at the detailed module level; on another, just at the system level. In some instances, the need for reliability measurement and prediction may dictate that only software errors of a specific type or classification are needed in the analysis. The essential message here is that the amount and detail of data required for analysis by the model is very heavily dependent upon the reliability information needs of the project manager, customer, or user.

In general however, whenever the model is applied to estimate 2 ( $N$  and  $q$ ) or 3 ( $N$ ,  $q$ , and  $\alpha$ ) parameters for a given module, subsystem, or system, four basis elements of data will be required. Table 5.1-1 summarizes these data requirements. A detailed discussion of each of these data elements follows.

##### 5.1.1 Test Occasion Date or ID

Although this data element may appear as self-explanatory, a brief discussion is needed. As far as the model is concerned, a test occasion may be a single computer run, a series of test runs executed in one day, a week, or even a month. In fact, it is not even necessary that the occasion involve computer runs. It may just denote the event that a certain length of time was devoted to visually inspecting the program code. Test occasion dates or identifiers should clearly denote the

Table 5.1-1. HISTORICAL DATA REQUIRED FOR INPUT TO MODEL

The following data are required by occasion for each module, subsystem, or system being analyzed:

<u>Data Element</u>	<u>Description</u>
(1) Test Occasion Date or ID ( $i$ )	A unique date (e.g., 9/79, 10/79...) or appropriate identifier (e.g. 1,2, 3...) which classifies errors by their actual or relative sequence of occurrence in time.
(2) Software Errors ( $n_i$ )	The total number of software errors detected on this occasion.
(3) Test Effort ( $t_i$ )	A relative measure of the total test effort expended on this occasion to detect the software errors indicated.
(4) Weighting Factor ( $w_i$ )	A relative measure of the percentage of total system source lines of code that were under test during this occasion. ( $0 < w_i \leq 1$ )

historical time sequence when the errors were detected or observed, rather than recorded. Lastly, the definition of "occasion" should be consistent from one occasion to the next.

#### 5.1.2 Software Errors

The theory upon which the models are based is concerned with explaining software error history characteristics in the real world of software development and testing. A software error for our purposes is defined as:

Software Error - any one of a number of different types of errors whose cause can be directly attributed to the activity of computer programming and that can be corrected by a change to the software itself.

According to this definition, software errors can be distinguished from software problem reports and software changes. Figure 5.1.2-1 is presented to aid in making clear how the relationships among these types of data are viewed.

In Figure 5.1.2-1 software problem reports (also called software failures) are viewed as symptoms or manifestations of latent errors in the system. The term "error" used here refers to the fact that the real cause of the problem could be something other than a valid software error. A time delay is usually associated with classifying the true cause of the problem. If a software error is the cause, one or more changes to the software will be needed to correct the error(s). Here too, a time delay occurs from the time that the problem is reliably classified as due to software to the time the change is made to correct the error.

Often software changes are likely to correct at least one, and often more than one, software error. In contrast, the fact that a software change





was made does not necessarily imply that a software error is corrected by this change. For example, our experience has indicated that at times software changes are sometimes made to work-around hardware problems.

The requirements for the reliability model are reflected by the dashed line in Figure 5.1.2-1. Narrowing the analysis to these categories does create problems; mainly due to time delays. At any given time there are unresolved problem reports. If this number is large, the use of the model is questionable. If the number is small, an estimate of the number of problem reports that are likely to be caused by software errors should be made.

One conclusion is obvious: the number of software errors can only be approximated using either failures or software changes. Reliability, defined in terms of the probability of a software error manifesting itself, requires software errors for its estimation. One reason for analyzing failures or changes, however, is that either might be quite useful in predicting errors, e.g., if failures or changes were found to be consistently proportional to errors over each test occasion. The validity of the model in estimating problem reports from historical problem reports has not been fully explored.

It is desirable that software errors be distinguished as such by the user via software error classification schemes implemented during the error data collection and correction process. A fundamental issue which indicates the need for this requirement is that unless software errors can be classified according to type and/or severity/criticality, then it must be assumed that all latent software errors have equal probabilities of manifestation. The validity of this assumption appears questionable.

### 5.1.3 Test Effort

Under ideal conditions of testing and error data collection, it would be desirable to record the amount of testing effort required to detect each software error. In reality, however, testing procedures are directed at examining the performance of the system as a whole or evaluating selected sets of functions, which could comprise a mixture of subsystems, modules, or units; all of this being heavily dependent on the phase of testing that is underway at a given time.

One critical data requirement is that some objective measure of the testing effort by function, system, module, etc., be provided by testing occasion for whatever level of the software system was under test for that occasion. One desirable objective measure of this testing effort is CPU time expended to perform the test. Another less precise measure could be actual wall clock hours expended to perform the test.

One important assumption made for test effort is that one occasion be comparable to every other occasion in terms of the type of testing expended in detecting errors. It is not required that the amount of time between testing occasions be equal. However, there should be no variation in the method of error detection applied on each occasion.

Recent data analysis results using the software reliability models described in Section 2 have shown that having a meaningful and accurate measure of the testing effort for each occasion is critical for obtaining valid reliability estimates and predictions. One finding that has been established regarding the testing effort measure is that absolute accuracy is not mandatory; relative accuracy will suffice. The implication of this finding is that estimates of testing effort may be used if they can be estimated accurately in a relative sense. Considering the time for the first testing occasion as unit time and expressing subsequent times as a multiple of this time would be adequate.

#### 5.1.4 Weighting Factor

The reliability model has been developed recognizing that the number of software errors detected on any occasion is a function of how much of the system was tested on that occasion. The model, therefore, does allow for testing situations wherein varying portions of the system are tested from one occasion to the next. For system reliability assessment, a data requirement in these cases is a count of the number of source lines of code under test and the estimated total source lines in the final system. The ratio of source lines under test to total source lines in the system is used as the weighting factor.

When testing is being done at the module or subsystem level, it is necessary to assign a weighting factor for each module. In this case, the modules tested must be identified along with the number of source lines of code in each. The weighting factors for each module are defined as the ratio of the size of the module to that of the total size of the system.

In lieu of a measure of module or system size measured in terms of source lines of code, an alternative metric is the number of words of core used by the modules tested on each occasion relative to the total words of core for the system. This later measure becomes useful for prediction purposes as a measure of relative size of the system under test, if the ratio of words of core required for a given source line of code is known or can be estimated.

#### 5.2 DATA ASSUMPTION

The software error data collected for model analysis is assumed to be taken from a system that is stable, i.e., undergoing no significant change in size, during the testing phase. This assumption implies that

the system from which error and test data are collected is not significantly changing from one test occasion to the next, in terms of lines of code being added, deleted, or modified, with the exception of changes needed to correct errors.

The rationale for this assumption is based on the fact that the model is designed to analyze data representing errors for one system, and not one set of data representing errors from multiple builds of a system. A new system build is defined as an update or functional enhancement made to a previous system version requiring the addition of a significant amount of new code.

Determining whether or not an amount of new code to be added is significant requires reasonable judgement on the part of the user of the model. The real question to ask in this case is when does a change to the system significantly affect the latent software error content of that system. For each new system build it is recommended that a new set of error data be collected and prior error history not used when estimating model parameters for the new system.

In many software development environments, the conditions required to meet this assumption of stability in the software system would only occur after unit testing has been completed, and would remain true throughout the duration of formal testing and integration and into the acceptance and operational testing phases.

One final implication of this assumption is that the error data that results from overlapping or new system builds be identified and distinguished as to which system they belong.

## SECTION 6

### SOFTWARE RELIABILITY REQUIREMENTS SPECIFICATION AND MEASUREMENT

#### 6.1 INTRODUCTION

The concept of software reliability has many interpretations, usually subjective and not very well-defined. In general, it is believed that reliability should somehow reflect satisfactory performance and freedom from software errors or malfunctions. In trying to incorporate these features into a definition of reliability, problems exist in defining the terms which are used to define reliability.

Perhaps the most serious problem in formulating an adequate definition is that these features are application dependent. Satisfactory performance, and to some extent even error free code, is in the eyes of the user. What is satisfactory for one user will be quite unsatisfactory for another one. With ten users there may be no two of them agreeing. With regard to errors in the software, some users would not be interested in the number of errors, but rather in the probability of their occurrence and their impact on the system when they do occur.

In order to serve as a useful measure of performance, a quality, such as reliability, must be defined in such a way that the method of performance evaluation is implicit in the definition. To impose a requirement for a certain level of performance, the statement of requirement must have an explicit statement of the method of evaluation, or at least strongly indicate the method.

This section attempts to address one aspect of software quality, namely reliability. A definition of reliability is given, a statement of the reliability requirement consistent with this definition is given, and a method of evaluating the results of a performance test is specified.

## 6.2 DEFINITION OF RELIABILITY

The user is, of course, ultimately concerned with how the software system will perform in an operational environment. Thus, he should design the performance test so that it represents that environment as closely as possible. Since any test is only a sample of some larger universe of all possible environmental demands, it is necessary to use statistical estimates derived from the test results to estimate the software system's reliability. These estimates are subject to variation due to two sources of primary concern. One is that the test is not representative of the environment, and the effects from this source can be minimized by careful design of the test. The other is that the test results are not representative of the true performance of the system, and this effect can be minimized, or at least estimated, by the use of statistical probability distributions.

To facilitate the application of a statistical distribution, we define software reliability then as:

The probability that no (or no more than a specified number of) software errors of a given type will occur during a designated time interval under specified testing conditions.

In this definition a number of key words, requiring further elaboration, are underlined.

- specified number - in a large system the probability of no errors is likely to be so low as to be meaningless; therefore, the definition is expanded to allow up to a reasonable number.
- software error - an area for negotiation. Some apparent system failures cannot be resolved as to whether they

are caused by the hardware, software, or human operator. In addition failures attributed to the software may have been caused by an error in requirements, design, or coding. It is critical that agreement be reached, preferably before work begins, on these areas.

- type of software error - agreement needs to be reached, again preferably before the start of work, on the types of errors to be counted. A separate requirement might be stated, e.g., for errors characterized as follows:

Type 1 - an error requiring a re-start of the system.

Type 2 - an error causing an erroneous or missing output in more than one application subsystem.

Type 3 - an error requiring the use of automatic recovery and causing an erroneous output in one application subsystem.

Type 4 - an error causing an output deviating from requirements more than a specified amount.

- time interval - the length of time for the operational test should be specified. An agreement should be reached as to whether down-time is to be allowed, and what to do about errors (i.e., repair them, patch-around, etc.) with regard to downtime.
- testing conditions - the specification of the environment for testing, e.g., inputs to the system and scenarios for human operators.

### 6.3 STATING RELIABILITY REQUIREMENTS

For each of the four types of errors discussed in the preceding section the reliability requirements are stated in the following form:

The probability that no more than  $x_1$  software errors of type  $x_2$  occurs during  $x_3$  intervals within a  $x_4$  operational test shall be at least  $x_5$ , where:

$x_1$  = number of errors allowed ( $x_1$  may be zero, where applicable/ needed)

$x_2$  = type of error (error types as stated above or others)

$x_3$  = length of interval (hours, days) for which  $x_1$  allowed.

$x_4$  = length of operational test interval (hours, days); should exceed  $x_3$ .

$x_5$  = required probability ( $0 < p < 1$ ), i.e., the specified reliability requirement.

Thus we allow for a different requirement for each error type, or for different combinations of types. The length of the operational test interval ( $x_4$ ) should exceed  $x_3$  so that we will have several intervals of length  $x_3$  for calculating the probability.



#### 6.4 MEASUREMENT OF PERFORMANCE

The Poisson distribution (or its continuous counterpart, the exponential) is appropriate for estimating the performance level for two very good reasons:

- The Poisson does not require an estimate of the population of errors (i.e., the number of latent errors of a particular type).
- The distribution of software errors for fixed time intervals has been demonstrated to be skewed. Most time intervals have errors occurring with a low frequency; still some few intervals have relatively high numbers of errors. The Poisson is a very good distribution for such events.

The probability density for the Poisson distribution is:

$$P(x;\mu) = \frac{\mu^x e^{-\mu}}{x!}$$

where  $\mu$  is the expected value of  $x$ .

In actual application, we can translate the requirement statement of the preceding section into a formula as follows:

$$\left[ P(x \leq x_1) = \sum_{x=0}^{x_1} \frac{\mu^x e^{-\mu}}{x!} \right] \geq x_5 \quad (6.1)$$

This formula should be applied for each of the types of error ( $x_2$ ). The value of  $\mu$  is to be obtained from multiplying the expected errors per unit time ( $n_i/x_4$ ) by the number of time units ( $x_3$ ) in the test interval. We have finally:

$$\left[ P(x \leq x_1) = \sum_{x=0}^{x_1} \frac{\left( \frac{x_3 n_i}{x_4} \right)^x}{x!} \exp \left( - \frac{x_3 n_i}{x_4} \right) \right]_{x_5} \quad (6.2)$$

#### 6.4.1 Numerical Example

Suppose in an operational demonstration test of 240 hours ( $x_4$ ) we had the following requirements:

Error Type ( $x_2$ )	Error Limit ( $x_1$ )	Time Interval for $x_1$ ( $x_3$ )	Required Probability
1	0	24	.95
2	1	24	.90
3	3	24	.75
4	2	8	.75

Suppose further we complete the 240 hours of run time, resolve all the failures and the following errors result:

$x_2$	Frequency of Occurrence
1	2
2	3
3	24
4	40

Did we pass or fail each of the 4 tests?

For Type 1, we have:

$$P(x \leq 0) = e^{-\mu}$$

$$\mu = \frac{(2)(24)}{240} = .2 \text{ errors per 24-hour period}$$

$$P(x \leq 0) = e^{-.115} \cong .82$$

For Type 2, we have:

$$P(x \leq 1) = \sum_{x=0}^1 \frac{\mu^x e^{-\mu}}{x!}$$

$$\mu = \frac{(3)(24)}{240} = .3 \text{ errors per 24-hour period}$$

$$P(x \leq 1) = e^{-.3}(1 + .3) \cong .96$$

For Type 3, we have:

$$P(x \leq 3) = \sum_{x=0}^3 \frac{\mu^x e^{-\mu}}{x!}$$

$$\mu = \frac{(24)(24)}{240} = 2.4 \text{ errors per 24-hour period}$$

$$P(x \leq 3) = .09072 \left[ 1 + 2.4 + \frac{(2.4)^2}{2} + \frac{(2.4)^3}{6} \right] \cong .78$$

For Type 4, we have:

$$P(x \leq 2) = \sum_{x=0}^2 \frac{\mu^x e^{-\mu}}{x!}$$

$$\mu = \frac{(40)(8)}{240} \cong 1.33 \text{ errors per 24-hour period}$$

$$P(x \leq 2) \cong .26360 \left[ 1 + \frac{4}{3} + \frac{8}{9} \right] \cong .85$$

Since the obtained probabilities exceed that required for each of the four types with the exception of Type 1, we fail to meet this requirement for Type 1 and meet the remaining ones.

#### 6.4.2 Confidence Level Considerations

In specifying the confidence required ( $x_g$ ) in the probability obtained, i.e., the specified reliability in an operational test, we are imposing a severe requirement, especially when the confidence is as high as .90. This severity may be exactly what the user requires. However, from the developer's standpoint, and from the standpoint of economics, it may be desirable to lower the confidence required. At least, the developer and customer should be aware of just how severe the requirement actually is. This can be done with the definitions and measures proposed thus far.

To illustrate this point, the requirements for error type 1 are examined. For Type 1, an error limit of zero errors in a 24-hour operation was specified. The required probability was .95. The only way this requirement could have been met was to run the entire 240 hours with no error occurring. The actual occurrence was only 1 error per 120 hours of testing, which is .2 errors per 24-hours. Considering the 240 hours as 10 discrete intervals of 24 hours each, in 8 of these 10 intervals there were no errors, or in only 2 of the 10 intervals errors did occur. Is this

performance close enough to that required (excluding the probability requirement) to indicate a "pass" rather than "fail"?

If one considers this requirement as too severe, he might agree with the following. A requirement should be stated such that if a performer just barely meets the requirement, he should be equally likely to fail or to pass a demonstration test.

## SECTION 7

### APPLYING THE MODEL TO SOFTWARE DEVELOPMENT PROJECTS

In addition to providing estimates of  $N$ ,  $q$  (or  $\phi$ ), and  $\alpha$  for a given set of software, other important error and reliability statistics can be derived using these parameter values. Some of these include the incidence of future error occurrences, the reliability of the software at some future time, and the estimated time required to achieve a specified reliability. Also, on projects where reliability requirements have been specified, the model can be applied to assist management in the analysis of trade-offs in the cost of further testing of the system versus increases in reliability.

The purpose of this section is to provide the equations needed by management to derive the error prediction, future test time, and reliability statistics. An example of a trade-off study is also provided.

## 7.1 PREDICTIONS OF FUTURE ERRORS

Once estimates of  $N$ ,  $q$  (or  $\phi$  for the Poisson), and  $\alpha$  are obtained from the model for a given set of software tested over  $k$  occasions, predictions of future errors can be derived. At the system level for the binomial, Poisson, or least squares model version, the prediction equations are:

$$n'_{k+1} = \bar{N}_{k+1} q_{k+1} \quad (7.1)$$

$$\begin{aligned} n'_{k+2} &= (\bar{N}_{k+1} - \alpha n'_{k+1}) q_{k+2} \\ &\vdots \\ &\vdots \end{aligned}$$

$$n'_{k+l} = \bar{N}_{k+1} - \alpha \sum_{i=k+1}^{k+l-1} n'_i q_{k+l}$$

Where:

$\bar{N}_{k+1} = (N - \alpha N_k)$  is the estimated number of errors remaining after  $k$  test occasions

$N_k$  is the cumulative errors observed for the system during the  $k$  test occasions

$$\begin{aligned} q_{k+1} &= 1 - (1-q)^{t_{k+1}} \\ &\vdots \\ &\vdots \end{aligned}$$

$$q_{k+l} = 1 - (1-q)^{t_{k+l}}$$

$q$  (or  $\phi$ ) is the probability of detecting an individual error during a unit time interval. For the Poisson, the equations  $q_{k+1}, q_{k+2}, \dots$  would be replaced by  $\phi_{k+1}, \phi_{k+2}, \dots$ , etc.,

$$(\bar{N}_{k+1} - \alpha n'_{k+1})$$

:  
:  
:

$$\left( \bar{N}_{k+1} - \alpha \sum_{i=k+1}^{k+l-1} n'_i \right)$$

are the estimates of the number of remaining errors in the system on each of the future occasions;

$t_{k+1}, t_{k+2}, \dots, t_{k+l}$ , are estimates, specified by the user of future testing effort to be applied to the system; and

$n'_{k+1}, n'_{k+2}, \dots, n'_{k+l}$  are the expected values of software errors detected during future occasions.

At the module level the equations are the same, if  $N$ ,  $q$  (or  $\phi$ ), and  $\alpha$  have been estimated for each module. If  $N$ ,  $q$  (or  $\phi$ ), and  $\alpha$  have been estimated for the system using the lower level module data, then the weighting factor ( $w_j$ ) for each module must be considered. The form of the prediction equation for occasion  $k+1$  now becomes:

$$n_{k+1,j} = \bar{N}_{k+1,j} q_{k+1,j} \quad (7.2)$$



Where,

$$\bar{N}_{k+1,j} = (w_j N - \alpha N_{k,j})$$

$$q_{k+1,j} = 1 - (1-q)^{t_{k+1,j}}$$

Similar substitutions for  $n_{k+2} \dots$ ,  $\bar{N}_{k+2} \dots$ , and  $q_{k+2} \dots$  are made for the remaining future occasions.

## 7.2 CURRENT AND FUTURE RELIABILITY ESTIMATES

Consider that at the end of  $k$  testing occasions we wish to evaluate the reliability of the software system that is under test. Using our definition of reliability, let  $t_{k+1}$  be the time interval for the next test occasion during which the system will be evaluated against the requirement that no more than a specified number ( $M$ ) of software errors will occur. If no errors are allowed, i.e.,  $M=0$ , then the reliability estimate ( $R'$ ) for the system using the binomial solutions for  $N$ ,  $q$ , and  $\alpha$  is:

$$p(M=0) = R' = \binom{\bar{N}_{k+1}}{0} q_{k+1}^{(0)} (1-q_{k+1})^{(\bar{N}_{k+1}-0)} \quad (7.3)$$

Further simplification of (7.3) results in:

$$R' = (1-q)^{t_{k+1} \bar{N}_{k+1}} \quad (7.4)$$

If no more than a specified number ( $M$ ) of errors ( $x$ ) are allowed during the requirements test, then the reliability estimate becomes:

$$p(x \leq M) = R' = \sum_{x=0}^M \binom{\bar{N}_{k+1}}{x} q_{k+1}^{(x)} (1-q_{k+1})^{(\bar{N}_{k+1}-x)} \quad (7.5)$$

When the solutions for  $N$ ,  $\phi$ , and  $\alpha$  are obtained using the Poisson model, corresponding equations for  $R'$  can be derived and are given as follows:

Where  $M=0$ :

$$P(M=0) = R' = e^{-\bar{N}_{k+1} \phi_{k+1}} \quad (7.6)$$

When no more than a specified number (M) of errors (x) are allowed:

$$p(x \leq M) = R' = \sum_{x=0}^M e^{\frac{-\bar{N}_{k+1}\phi_{k+1}}{x!}} (\bar{N}_{k+1}\phi_{k+1})^x \quad (7.7)$$

At the module level, if  $N$ ,  $\phi$  and  $\alpha$  have been estimated for each module, the equations do not change. If  $N$ ,  $\phi$ , and  $\alpha$  are estimated for the system using the lower level module data, then substitutions for  $\bar{N}_{k+1}$ ,  $\phi_{k+1}$ , and  $t_{k+1}$  are needed, as follows, to apply to the individual module (j):

$$\bar{N}_{k+1,j} = (w_j N - \alpha N_{k,j})$$

$$\phi_{k+1,j} = 1 - (1-\phi)^{t_{k+1,j}}, \text{ and}$$

$t_{k+1,j}$  is substituted for  $t_{k+1}$

For future reliability estimates, consider that a requirement has been established which specifies that during some future testing occasion (denoted here as  $t_{k+2}$ ) the probability that no more than a specified number (M) of errors will occur is R. The project manager, in this case, needs to determine how much additional testing effort ( $t_{k+1}$ ) is needed, prior to the actual beginning of the future test interval ( $t_{k+2}$ ), to get the system up the specified reliability level (R).

At the system level, then, assume that  $n_{k+1}$  software errors will be detected during the interval  $t_{k+1}$ . From (7.4) for the binomial model, the specified reliability on occasion (k+2) is given by:

$$R_{k+2} = (1-q)^{t_{k+2}(\bar{N}_{k+1} - n_{k+1})}$$

which reduces to:

$$R_{k+2} = (1-q)^{t_{k+2}(\bar{N}_{k+1})(1-q)^{t_{k+1}}} \quad (7.8)$$

when (7.1) is substituted for  $n_{k+1}$ .

Solving now for  $t_{k+1}$ , we have:

$$t_{k+1} = \frac{\ln \left[ \frac{\ln R_{k+2}}{t_{k+2} \bar{N}_{k+1} \ln(1-q)} \right]}{\ln(1-q)} \quad (7.9)$$

After this interval ( $t_{k+1}$ ) of future testing, the probability that the number of errors occurring during a specified interval ( $t_{k+2}$ ) beyond  $t_{k+1}$  is less than or equal to a given amount (M) is:

$$R_{k+2} = p(x \leq M) = \sum_{x=0}^M \binom{\bar{N}_{k+1} - \alpha n_{k+1}}{x} q_{k+2}^x (1-q)^{t_{k+2}(\bar{N}_{k+1} - \alpha n_{k+1} - x)} \quad (7.10)$$

where:

$$q_{k+2} = 1 - (1-q)^{t_{k+2}}$$

For the Poisson system level model, the corresponding equations to (7.8) through (7.10) are:

$$R_{k+2} = e^{-(\bar{N}_{k+1} - \alpha n'_{k+1}) \phi_{k+2}} \quad (7.11)$$

where:

$$n'_{k+1} = \bar{N}_{k+1} (1 - (1-\phi)^{t_{k+1}})$$

Solving (7.11) for  $t_{k+1}$  gives:

$$t_{k+1} = \frac{\ln \left[ 1 - \left( \bar{N}_{k+1} + \frac{\ln R_{k+2}}{\phi_{k+2}} \right) / \alpha \bar{N}_{k+1} \right]}{\ln(1-\phi)} \quad (7.12)$$

and for the Poisson, the probability of passing a reliability requirements test is:

$$R_{k+2} = p(x \leq M) = \sum_{x=0}^M \frac{e^{-(\bar{N}_{k+1} - \alpha n'_{k+1}) \phi_{k+2}} \left[ (\bar{N}_{k+1} - \alpha n'_{k+1}) \phi_{k+2} \right]^x}{x!} \quad (7.13)$$

Applying (7.8) through (7.13) at the module level, assuming  $N$ ,  $q$  (or  $\phi$ ) and  $\alpha$  have been estimated for the system using the lower level module data, requires similar substitutions for the terms  $\bar{N}_{k+1}$ ,  $q_{k+1}$ ,  $q_{k+2}$ ,  $\phi_{k+1}$ ,  $\phi_{k+2}$ ,  $t_{k+1}$ , and  $t_{k+2}$ , as indicated earlier, including the substitution  $t_{k+2,j}$  for  $t_{k+2}$ .

It is important to note here that all the equations in this section are properly formulated under the assumption that ( $\alpha=1$ ) over the future test occasions. A more general formulation of these equations is needed, given the work described in Section 3.4, concerned with determining the time to achieve a specified number of errors remaining under imperfect conditions (i.e.,  $\alpha<1$ ) of error correction.

### 7.3 EXAMPLE MODEL APPLICATION TO TRADE-OFFS

Consider that the following reliability requirements\* have been specified for a software system composed of three subsystems (A, B, and C). The system is to be demonstrated during a reliability demonstration test.

- During a 48-hour continuous reliability test, the number of critical errors allowed for each subsystem are:

Subsystem A - a maximum of 1

Subsystem B - a maximum of 3

Subsystem C - a maximum of 13

Consider also that if the developer can met these requirements, the customer will award the following dollar amounts by subsystem:

<u>Subsystem</u>	<u>Award Amount</u>
A	100K
B	50K
C	25K

Addressing the problem directly, the developer needs to answer the question, "When we go into the 48-hour test, at what level of performance do the subsystems have to be operating in order to be reasonably certain of passing the test?"

\*Similar requirements could have been specified for one system, where the errors allowed were classified into three different categories of errors, classified by type, severity, criticality, etc.

In order to answer the question, the developer needs to test each of the subsystems during some time interval preceding the reliability test, such that the conditions of the 48-hour test are duplicated over repeated occasions during this interval. Of course, software errors of the type (in this example only critical errors are examined) that will be evaluated during the requirement test need to be collected during the occasions preceding the actual test. Test effort data is also collected during this test interval and the error rates are calculated for each subsystem.

Using a basic table of probabilities for the Poisson distribution function, it is found that to be reasonably certain (e.g., 90 percent probability level) of meeting the requirements, the performance level (i.e., error rate) of each subsystem should be:

<u>Subsystem</u>	<u>Errors/48-Hours</u>
A	0.5
B	1.8
C	9.5

Consider that the predicted error rates during the 48-hour test, obtained using the model, are 1.6 for Subsystem A, 3.6 for Subsystem B, and 12 for Subsystem C. By using these numbers as means of a Poisson distribution, the probability of passing the requirements test can be determined from a table of the Poisson distribution. The error rates and their corresponding probabilities are summarized below.

<u>Subsystem</u>	<u>Predicted Error Rate</u>	<u>Corresponding Probability</u>
A	1.6	.53
B	3.6	.52
C	12.0	.68



With no further testing prior to the 48-hour test, the expected award amount or payoff is:

$$\begin{aligned}\text{Expected amount} &= .53 (100K) + .52 (50K) + .68 (25K) \\ &= 96K\end{aligned}$$

The 96K should be compared with 175K which is the total award amount given that the requirements as stated are met.

The developer may now desire or attempt to improve the performance level of his system in such a way as to maximize his expected payoff from the customer, using the time remaining prior to the requirement test. Consider now that the developer sets out to reduce the error rates of each subsystem by 50 percent, and that this can be accomplished assuming a fixed testing effort ( $E_1$ ). The improved performance levels and probabilities that would result for each subsystem are as follows:

<u>Subsystem</u>	<u>New Performance Level</u>	<u>New Probability of Passing Test</u>
A	0.8	.81
B	1.6	.89
C	6.0	.99

For the developer, the expected payoff that would result from expending the effort ( $E_1$ ) on one subsystem versus another would be:

<u>Strategy*</u>	<u>New Expected Payoff</u>
1: Expend $E_1$ on A	.81 (100K) + .52 (50K) + .68 (25K) = 124K
2: Expend $E_1$ on B	.53 (100K) + .89 (50K) + .68 (25K) = 114.5K
3: Expend $E_1$ on C	.53 (100K) + .52 (50K) + .99 (25K) = 103.8K

\*More complex strategies could also apply here.

The benefit and cost of each strategy then would be as follows assuming that the cost of  $E_1$  is, for example, 20K:

<u>Strategy</u>	<u>Benefit</u>		<u>Cost</u>
1	124K - 96K = 28K	} LESS COST OF $E_1$	= 8K
2	114.5K - 96K = 18.5K		= -1.5K
3	103.8K - 96K = 7.8K		= -12.2K

Thus, given the developer's objectives (reduce error rates by 50 percent) and available resources (a fixed effort  $E_1$ ), strategy 1 should be chosen. For strategy 2 and 3, the cost of improvement ( $E_1$ ) exceeds the payoff. Under these conditions, the best strategy for subsystems B and C may simply be not to expend any further effort toward improving their performance levels.

## SECTION 8

### DEFINITION OF TERMS

This section provides definitions for terms used throughout this report within the context of software reliability predictions. Related terms, not necessarily used in the report, are also provided. Synonyms and/or quantitative definitions are included as appropriate. Where multiple definitions exist only the one(s) relevant to this report are given. The IBM Data Processing Glossary [11], and Joel D. Aron's text, The Program Development Process [1], served as supplementary sources for the definitions.

- Availability - The degree to which a system or resource is ready when needed to process data.
- Debugging - The process of detection, location, and removal of errors.
- Error - A condition that could produce a discrepancy between data processing results and true, or theoretically correct (from the user's point of view), results.
- Error Detection Probability ( $q$  or  $\phi$ ) - The probability of any one error being detected within a unit time interval. The reliability models presently developed assume that this error detection probability,  $q$  (or  $\phi$ ), is constant from one testing occasion to another, and is independent of the number of other errors detected on previous occasions.

- Error Rate - The number of errors which occur during a unit time interval. It may refer to either observed values, estimated values, or predicted values.
- Failure - A symptom or manifestation of an error. See also software problem reports.
- Module - A program unit that is discrete and identifiable with respect to compiling, combining with other units, and loading. For example, a module may be input to or output from an assembler, compiler, linkage editor, or executive routine. A module is the smallest (lowest level) element of software for which error data can be recorded. See Figure 8-1.
- Operating System - Software which controls the execution of computer programs and which may provide scheduling, debugging, input/output control, accounting, compilation, storage assignment, data management, and related services.
- Program - A sequence of instructions or steps that completely describes a procedure in order to achieve a specified result. In the context of this report computer program is implied; hence a program that can be executed on a computer to process data. See Figure 8-1.
- Project - An activity structured and organized to accomplish a specified task within financial and time constraints. A programming project has as its objective

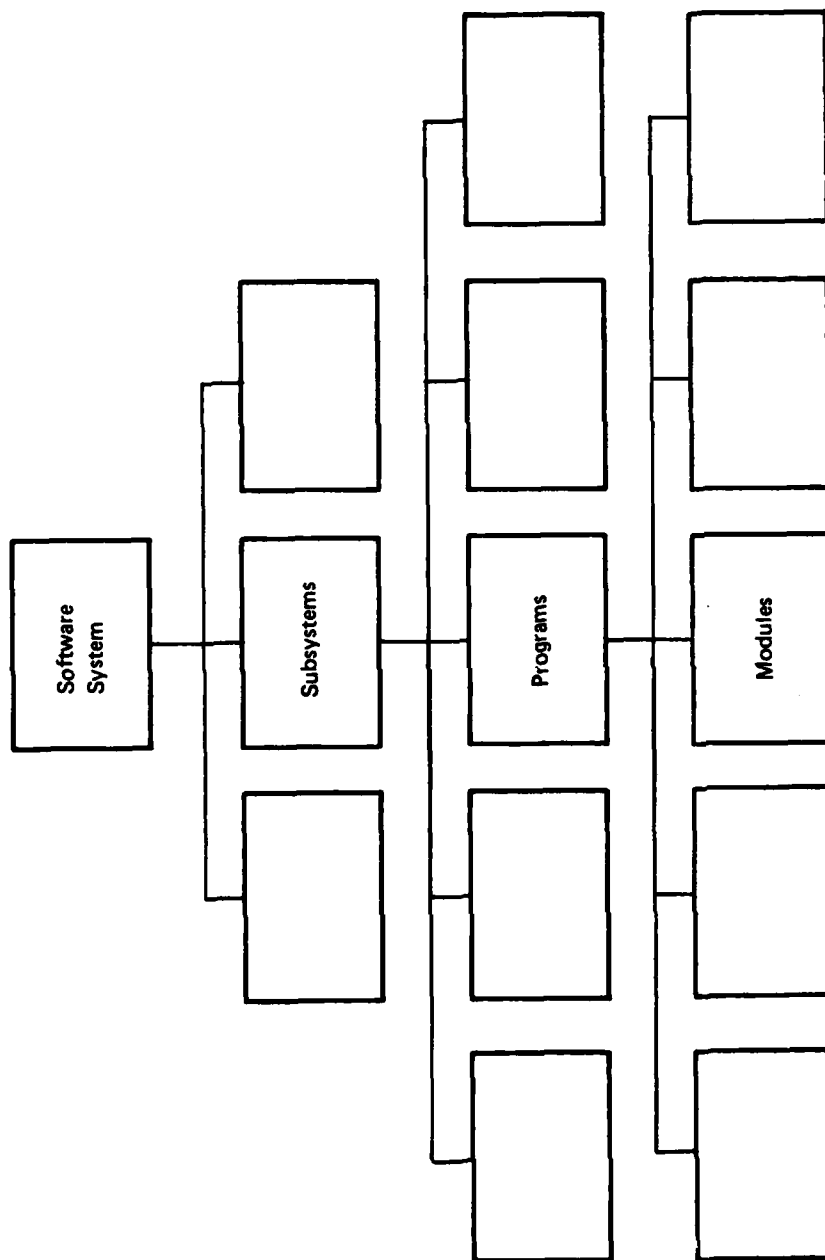


Figure 8-1. SOFTWARE SYSTEM HIERARCHY

the production of a program system. Frequently, a programming project is a subset of a larger systems project, where the systems project may include, in addition to the programming project, a hardware project, systems operations, integration and test, etc. Synonym: Program (commonly within the Federal government.)

- Software - In the broad sense, a set of programs (or program system(s)), procedures, and documentation. In the context of a software error, the reference is more strictly attributed to a set of programs and their associated errors.
- Subsystem - A logical part of a system that is itself a system, and is usually capable of operating independently of and/or asynchronously with its controlling system and other subsystems. See Figure 8-1.
- System - A collection of elements which are organized such that they satisfy a set of functional and performance specifications. In the context of this report, a system refers to a collection of computer programs. See Figure 8-1.
- Software Error - Any one of a number of errors that can be classified as being caused by or attributed to the activity of computer programming, and that can be corrected by a change to the software itself.

- Software Problem Reports (SPRs) - Reports written usually during formal system test which describe error conditions or system failures purported to be attributable to the software. In the context of this report SPRs are considered symptoms of latent errors in the software.
  
- Software Reliability - The probability that no (or no more than a specified number of) software errors of a given type will occur during a specified future time interval under specified testing conditions.
  
- Software System - A collection of modules which are computer programs and their related documentation. Typically, a software system includes an operating system and an applications system where the operating system provides certain support functions and the application system satisfies specific user data processing requirements.
  
- Test Occasion - An event of error data collection; each occasion should have a time interval associated with it, otherwise, the implication to the model is that all test occasions are of equal length of time. If the event is a computer run, then the CPU time should be used; if the event is a manual debugging, the elapsed time should be used. One additional important assumption made here is that one occasion be comparable to every other occasion in terms of the time spent (testing effort) in detecting errors. It

is not required that the amount of time between testing occasions be equal. However, there should be no variation in the method of detection.

- Unit - See definition of module.



## SECTION 9

### REFERENCES

- [1] Aron, J. D.  
The Program Development Process, The Individual Programmer.  
Reading, MA: Addison-Wesley Pub. Co. Inc., 1974.
- [2] Baker, W. F.  
Software Data Collection and Analysis, RADC-TR-77-192,  
Technical Report  
Rome Air Development Center, Air Force Systems Command  
Griffiss Air Force Base, New York, June 1977. (A041644)
- [3] Brooks, W. D., Weiler, P. W.  
Software Reliability Analysis, Technical Report, PCI 6G39.  
Gaithersburg, MD: IBM Corporation, FSD, December 1976.
- [4] Brooks, W. D., Weiler, P. W.  
Software Reliability Analysis, IBM Technical Report, FSD 77-0009.  
Gaithersburg, MD: IBM Corporation, FSD, February 1977.
- [5] Brooks, W. D., Kocher, D. F., Kitley, R. W., Weiler, P. W.  
Software Reliability Analysis, Technical Report, PCI 7G39.  
Gaithersburg, MD: IBM Corporation, FSD, December 1977.
- [6] Dodes, I. A.  
Numerical Analysis for Computer Science.  
New York, NY: Elsevier North-Holland, Inc., 1978.
- [7] Fries, M. J.  
Software Error Data Acquisition, RADC-TR-77-130, Final Technical  
Report, Boeing Aerospace Co., April 1977, (A039916)
- [8] Goel, A. L., Okumoto, K.  
Software Failure Analysis by Non-Homogeneous Poisson Process,  
Syracuse, NY: Syracuse University (Unpublished paper).
- [9] Goel, A. L.  
A Software Error Detection Model with Applications.  
Syracuse, NY: Syracuse University, 1979 (Unpublished paper).

- [10] Hildebrand, F. B.  
Introduction to Numerical Analysis. New York, NY:  
McGraw-Hill Book Company, 1956.
- [11] IBM Corporation.  
Data Processing Glossary, Publication No. GC20-1699.  
White Plains, NY: IBM Corporation, 1977.
- [12] Jelinski Z. and Moranda, P. B.  
"Applications of a Probability-Based Model to a Code Reading Experiment,"  
Proceedings, IEEE Symposium on Computer Software Reliability, April  
30 through May 2, 1973, p. 78.
- [13] Motley, R. W.  
Software Reliability Analysis, Technical Report, PCI 8G39.  
Gaithersburg, MD: IBM Corporation, FSD, September 1978.
- [14] Motley, R. W. and Brooks, W. D.  
Statistical Prediction of Programming Errors, RADC-TR-77-175,  
IBM Corp., Final Technical Report, May 1977. (A041106)
- [15] Motley, R. W., Brooks, W. D.  
Software Reliability Model: Usability and Acceptability  
in the Project Environment. Technical Report, OHE Project.  
Gaithersburg, MD: IBM Corporation, FSD, December 1979.
- [16] Shaefer, R. E. et al.  
Validation of Software Reliability Models, RADC-TR-79-147,  
Final Technical Report, Hughes Aircraft Co., June 1979. (A072113)
- [17] Shoeman M. L.  
"Operational Testing and Software Reliability Estimation During  
Program Development, " Record, 1973 IEEE Symposium  
on Computer Software Reliability, April 30 through May 2, 1973,  
IEEE Catalog No. 73, CH0741-9CSR, pp. 51-57.
- [18] Sukert, Captain Alan N.  
A Software Reliability Modeling Study, RADC-TR-76-247,  
In-house Report,  
Rome Air Development Center, Air Force Systems Command,  
Griffiss Air Force Base, New York, August 1976. (A030437)

- [19] Thayer, T. A., et al.  
Software Reliability Study, RADC-TR-76-238, Final Technical Report,  
TRW Systems Group, August 1976. (A030798)
- [20] Willman, H. E., et al.  
Software Systems Reliability: A Raytheon Project History,  
RADC-TR-77-188, Final Technical Report, Raytheon Co., June 1977.  
(A040992)

## **MISSION of Rome Air Development Center**

*RADC plans and executes research, development, test and selected acquisition programs in support of Command, Control Communications and Intelligence (C<sup>3</sup>I) activities. Technical and engineering support within areas of technical competence is provided to ESD Program Offices (POs) and other ESD elements. The principal technical mission areas are communications, electromagnetic guidance and control, surveillance of ground and aerospace objects, intelligence data collection and handling, information system technology, ionospheric propagation, solid state sciences, microwave physics and electronic reliability, maintainability and compatibility.*